Universidad Zaragoza
1542

Proyecto Fin de Carrera

Desarrollo de un sistema de seguimiento de usuarios con iPhone para visualizarlos en un modelo 3D

Autor/es

Jorge Pérez Lahera

Director/es y/o ponente

Ana Cristina Murillo Arnal
Sune Lehmann Jørgensen
Michael Frederiksen

Escuela de Ingeniería y Arquitectura
2012

Repositorio de la Universidad de Zaragoza – Zaguan
http://zaguan.unizar.es

Desarrollo de un sistema de seguimiento de usuarios con iPhone para visualizarlos en un modelo 3D

RESUMEN

El objetivo de este proyecto es desarrollar un sistema de seguimiento de usuarios con un iPhone y un modelo 3D del campus de la Technical University of Denmark. El usuario podrá activar el seguimiento tras abrir una aplicación en el iPhone siempre y cuando se encuentre en alguna de las áreas donde haya un modelo 3D disponible. Los usuarios que hayan activado el seguimiento serán mostrados en estos modelos 3D en forma de avatares. Los modelos 3D junto con los avatares pueden ser visualizados usando cualquier navegador de escritorio en la página web realsite.dk.

Los sensores GPS de los Smartphones no son normalmente muy precisos. Para desarrollar buenos algoritmos en el sistema de seguimiento requerido, la precisión de este sensor tiene que ser analizada. Por esta razón el proyecto empieza con un extenso estudio de la precisión de los sistemas de localización en el iPhone y de los parámetros que pueden configurarse. Se estudian tanto posiciones fijas como en movimiento. Este estudio revela que el error medio en posiciones estáticas es en torno a 8 metros y bastante mayor para las posiciones en movimiento. Sin embargo es muy rápido determinando la primera posición lo cual lo hace en menos de 10 segundos en la mayoría de los casos. Utilizando los resultados de este estudio, se han diseñado varios filtros para eliminar las posiciones menos precisas. Además, también se ha desarrollado una técnica que permite detectar cuando el usuario entra dentro de un edificio sin usar ninguna información adicional más que la que los servicios de localización ofrecen.

Las dos partes mas importantes de este sistema han sido desarrolladas en su totalidad en este proyecto fin de carrera. Estas son una aplicación para el sistema operativo móvil iOS y un algoritmo para representar a los avatares de los usuarios en los modelos 3D. La aplicación recoge las posiciones de los usuarios, utilizando el GPS del dispositivo, las filtra, las guarda y las manda a un servidor de internet donde son almacenadas en una base de datos. También permite visualizar las sesiones anteriores en las que el seguimiento ha sido activado y tomar una foto que será utilizada en el avatar del usuario. La representación de los avatares en el modelo no se puede llevar a cabo con las posiciones que el dispositivo iOS obtiene ya que no son suficientemente precisas. Por lo que se diseñó un algoritmo que genera a partir de las posiciones GPS recibidas una ruta realista, factible y libre de obstáculos en el modelo. Un detalle importante por ejemplo, es que hace que los avatares utilicen escaleras y puertas de edificios cuando se detecta que han cambiado de altitud o entrado a un edificio respectivamente.

Índice general

Capítulo 1

Introducción

Este proyecto fin de carrera fue llevado a cabo en la empresa Utopian city_scape (UCS), Copenhague, como parte de un acuerdo de colaboración con la Technical University of Denmark (DTU). El alumno de la EINA era estudiante de intercambio Erasmus en la DTU durante el curso 2011 – 2012 durante el que realizó este proyecto. UCS ha estado desarrollando modelos 3D de complejos de edificios durante los últimos cinco años. Ejemplos de estos son los modelos del campus de la DTU o de la fábrica de Carlsberg. El propósito de esta empresa es el desarrollar modelos 3D interactivos usando tecnología de videojuegos. Todos estos modelos 3D son accesibles usando la página web de la compañía (realsite.dk).

Pero ahora UCS quiere dar un paso adelante conectando estos modelos con el mundo real aprovechando la popularización de los Smartphones. El objetivo de este proyecto fin de carrera es desarrollar un sistema que obtendrá las posiciones de los usuarios y los representará en el modelo 3D del área correspondiente. Todos las pruebas desarrolladas en este proyecto han sido desarrolladas en el área del campus de la DTU ya que UCS ha desarrollado un modelo 3D muy preciso de ella.

1.1. Motivación

2011 fue el primer año en la historia en el cual se distribuyeron más smartphones que PCs[1]. De acuerdo con este estudio llevado a cabo por Catalyst, el número de smartphones y tablets distribuidos fue de 488 millones mientras que el de portátiles y ordenadores de escritorios fue de 414 millones. Estos dispositivos contienen un gran número de sensores como GPS, acelerómetro y brújula digital. Además, permiten a los usuarios estar conectados a internet de forma continua utilizando las redes celulares. Un estudio del mercado de los smartphones puede encontrarse en el apéndice D capítulo 1.1.

La proliferación de estos smartphones esta cambiando la forma en la que los usuarios se conectan a internet, nuevos servicios han sido creados utilizando estas plataformas en los últimos años. Entre estos servicios uno de los más populares son los servicios basados en localización o LBS de sus siglas en inglés. Ejemplos de estos son la redes sociales Foursquare[2] o Facebook places[3], las cuales permiten a los usuarios compartir su localización con sus amigos. Otros ejemplos de estos servicios son las aplicaciones (llamadas apps en este documento) de seguimiento de usuarios, las cuales son similares al sistema desarrollado en este proyecto fin de carrera como son Endomondo[4], Runkeeper[5], Runtastic[6] y Glympe[7]. Mientras que las tres primeras sirven para monitorizar la actividad deportiva. La última sirve para compartir la posición del usuario en tiempo real.

Sin embargo, el sistema desarrollado en este proyecto, a diferencia de los sistemas de seguimiento descritos anteriormente y de la mayoría que se encuentran en la actualidad en el mercado no utiliza un mapa plano para representar a los usuarios. Usa un modelo 3D de alta precisión del área en el que se llevo a cabo el seguimiento del usuario. Esto hace que este sistema sea tan innovador que no se han encontrado sistemas similares en el mercado. Aunque la mayoría de mapas electrónicos que son normalmente utilizados en ordenadores o Smartphones son planos como Google Maps o OpenStreetMap, el uso de mapas 3D se esta convirtiendo más popular en los últimos años. Google Maps tiene mapas 3D de las ciudades más importantes del mundo. Además otras compañías como Upnext[8] producen mapas 3D de ciudades. El pasado junio, Apple anunció que la nueva versión de iOS, iOS 6, introducirá unos nuevos mapas 3D desarrollados por la

[1] http://www.engadget.com/2012/02/03/canalys-more-smartphones-than-pcs-shipped-in-2011/
[2] https://foursquare.com/
[3] http://www.facebook.com/about/location/
[4] http://www.endomondo.com/login
[5] http://runkeeper.com/
[6] http://www.runtastic.com/
[7] http://glympse.com/
[8] http://upnext.com

propia compañía. Un ejemplo de estos mapas 3D se muestra en la Figura 1.1. Todos estos movimientos indican que los mapas 3D son el futuro de los mapas digitales.

Figura 1.1: Nuevos mapas 3D en iOS 6[a]

[a] `http://www.apple.com/ios/ios6/#maps`

UCS ha estado desarrollando modelos 3D durante más de cinco años. El seguimiento de usuarios usando estos modelos 3D producirá una novedosa representación de la realidad que nunca ha sido desarrollada anteriormente. Este proyecto fin de carrera es una prueba de este concepto. El uso de modelos 3D precisos añade información adicional al seguimiento de usuarios que puede ser utilizada para aumentar la precisión como es demostrado en este proyecto. El sistema puede diferenciar donde están los edificios y el terreno, por lo tanto el usuario puede ser colocado en cada momento en el lugar correcto. Otra información que puede ser fácilmente obtenida es la altura de cualquier punto en el modelo, la pendiente de una superficie, la localización de las carreteras, las puertas de los edificios y las escaleras etc. Toda esta información junto con algoritmos inteligentemente diseñados puede transformar rutas muy imprecisas en rutas creíbles que pueden ser representadas en un modelo 3D de alta precisión.

1.2. Colaboración con Utopian City_Scape

Como se comento al principio de la introducción este proyecto fin de carrera se desarrollo en colaboración con la empresa Utopian City_Scape. UCS expuso al principio del proyecto los requerimientos generales de este. Estos requerimientos eran muy generales dando lugar a la innovación del alumno. El objetivo durante todo el proyecto ha sido encontrar algo útil e interesante para ambas partes, la empresa y el estudiante. Durante todo el proyecto el estudiante ha colaborado con UCS. Ellos supervisaban y guiaban el trabajo del estudiante, dejando para este todo el desarrollo.

1.3. Requerimientos del proyecto

El objetivo del proyecto es desarrollar un sistema para el seguimiento de usuarios en el área del campus de la DTU para localizarlos en un modelo 3D. El sistema tiene que ser fácilmente escalable a otros modelos 3D de la compañía e integrable en sus productos actuales. Los módulos que se deben desarrollar en este proyecto son:

- Análisis de los sistemas de localización en iOS.
- iOS app para adquirir la localización de los usuarios y mandarla al servidor de la empresa.
- Servidor que guarde, procese y mande las localizaciones al reproductor de los modelos 3D.
- Reproductor para los modelos 3D que permita ver los avatares de los usuarios, el cual incluye algoritmos para mejorar la ruta recibida del dispositivo iOS.

1.4. Estructura de esta memoria

Capítulo 2 es un estudio de los sistemas de localización en iOS. El análisis incluye no solo posiciones estáticas pero también posiciones en movimiento.

Capítulo 3 muestra los algoritmos que fueron desarrollados para representar a los avatares de los usuarios en el modelo. Además las rutas resultado son analizadas y comparadas con las originales.

Capítulo 4 presenta el sistema final integrado con el servidor web de la empresa en realsite.dk.

Capítulo 5 resume toda la memoria enfatizando los resultados más importantes.

Capítulo 2

Estudio de la precisión de los servicios de localización en iOS

Como se va a comprobar en este capítulo, los sistemas de localización en la mayoría de los smartphones no se caracterizan por su alta precisión. Son rápidos y están disponibles casi en el 100 % de las localizaciones pero tienen errores medios de varios metros. Esto hace que para aplicaciones como la que se desarrolla en este proyecto no se puedan utilizar los datos procedentes del dispositivo sin ninguna modificación. Por lo tanto un estudio de la precisión de estos servicios de localización como el que se desarrolla en este capítulo es esencial para desarrollar los algoritmos necesarios para transformar estas posiciones en una ruta representable.

2.1. Introducción

Todos los smartphones modernos utilizan un sistema de localización híbrido. Es decir, utilizan tres tecnologías diferentes para determinar la posición de los usuarios. Estas son posicionamiento por satélite (A-GPS), posicionamiento Wi-Fi y posicionamiento celular. La utilización de estas tres tecnologías permite

obtener una posición en casi el 100 % de los emplazamientos a parte de permitir diferentes niveles de precisión y de consumo de batería.

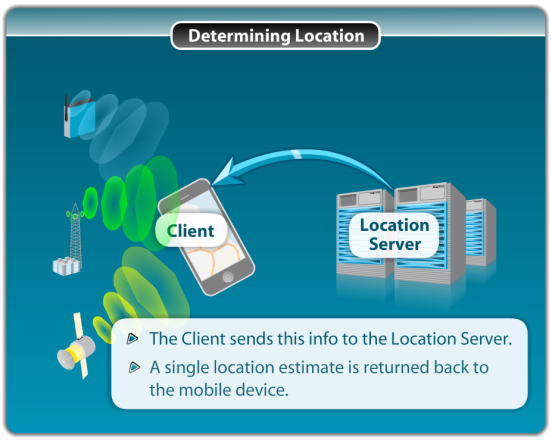


Figura 2.1: Ejemplo del sistema de posicionamiento hibrido[a]

[a] `http://www.skyhookwireless.com/howitworks/loader_howitworks.swf`

El sistema más preciso de los tres es posicionamiento por satélite. La mayoría de los smartphones que contienen un sensor GPS, utilizan una tecnología que se llama Assisted GPS (A-GPS) [GS05]. Esta tecnología permite que un servidor GPS al que el dispositivo accede a través de internet desarrolla muchas de las funciones. Esto mejora el consumo de batería y el time-to-first-fix [ML08] sobre los receptores GPS tradicionales [Zan09]. Los errores en la propagación de las señales procedentes de los satélites limita la precisión del GPS. Estos errores son los típicos errores que aparecen en todas las comunicaciones por satélite. La señales no se propagan a velocidad constante lo que produce retardos inconsistentes. Además, las señales chocan con obstáculos antes de llegar a la antena del dispositivo lo que produce degradación debido a la propagación multicamino. Esta es una de las principales razones por las que el GPS no funciona bien en áreas urbanas y en interiores.

No tan preciso como el GPS en exteriores pero mucho más preciso en interiores y en determinadas áreas urbanas tenemos el posicionamiento utilizando redes Wi-Fi. Actualmente es difícil encontrar un lugar dentro de una ciudad donde no hay Wi-Fi APs (Acess points) disponibles, un estudio de la densidad de APs puede encontrarse en [JL07]. El sistema más utilizado de esta tecnología se llama fingerprinting y se basa en comparar patrones de señales Wi-Fi, los cuales son obtenidos manualmente junto a su localización, mas similar a la huella de redes Wi-Fi que el dispositivo esta midiendo [Hub]. Hay varias tecnologías desarrolladas para esta comparación, la mas utilizada es K-nearest neighbor estimation

[Zan09] [SC05]. El sistema menos preciso pero el que más disponibilidad tiene y el que menos consumo energético es el sistema de posicionamiento celular. Este utiliza las señales de las redes 2G y 3G para determinar la posición de los usuarios.

iOS proporciona el llamado Core Location Framework, que permite a los desarrolladores obtener la localización de los tan solo configurando unos pocos parámetros. Es necesario crear una instancia de la clase *CLLocationManager* y configurar las propiedades *desiredAccuracy* y *distanceFilter*. Una vez esta hecho esto hay que asignar un delegado que implemente el protocolo *CLLocationManagerDelegate Protocol*[1] y llamar al método *startUpdatingLocation* para empezar a recibir las localizaciones del usuario. La propiedad *desiredAccuracy* permite configurar la precisión deseada en las localizaciones aunque no la garantiza, su comportamiento es estudiado en la sección 2.3.1. Las localizaciones se reciben en forma de instancias del objeto *CLLocation* el cual tiene cinco propiedades diferentes *coordinate, altitude, horizontalAccuracy, verticalAccuracy y timestamp.* Los nombres expresan lo que contienen pos si solos por lo que no necesitan explicaciones

2.2. Configuración experimental

Los dispositivos estudiados en este proyecto son un iPhone 3GS y un reloj Garmin forerunner 405. El estudio es llevado a cabo principalmente sobre el iPhone ya que es el dispositivo sobre el que se desarrolla el sistema utilizando el Garmin como comparación de otro dispositivo móvil más preciso.

Investigaciones anteriores en esta área colocan los dispositivos en un trípode y orientan la dirección de máxima radiación de la antena GPS hacia el cielo. Ejemplos de ello pueden encontrarse en [Zan09] y [ZB11]. Esto no es una posición realista si pensamos en el uso normal de un Smartphone. Por lo que en este estudio el iPhone se ha colocado en el bolsillo de un pantalón vaquero y el reloj en la muñeca de la mano izquierda. En el estudio de posiciones estáticas las localizaciones fueron obtenidas en periodos de dos minutos. 1887 posiciones fueron obtenidas para el estudio de la precisión en posiciones estáticas y 372 para la comparación entre los dos dispositivos. Mientras que para las posiciones en movimiento 1791 posiciones fueron obtenidas con el iPhone y 376 con el Garmin. Todos los estudios expuesto en esta sección se llevaron a cabo en el área del campus de la Technical University of Denmark.

[1] CLLocationManagerDelegate Protocol Reference `https://developer.apple.com/library/ios/#documentation/CoreLocation/Reference/CLLocationManagerDelegate_Protocol/CLLocationManagerDelegate/CLLocationManagerDelegate.html`

En el iPhone, los datos fueron obtenidos utilizando una aplicación que fue desarrollada completamente como parte de este proyecto fin de carrera. Esta aplicación permite empezar y terminar el seguimiento, al mismo tiempo que guarda todas las posiciones recibidas en una base de datos en la memoria interna del teléfono. También permite observar las sesiones que están guardadas en la base de datos en un mapa y exportarlas al ordenador o enviarlas al servidor de UCS. Parte del código desarrollado en esta aplicación fue utilizado en el desarrollo de la aplicación para el sistema final.

El Garmin permite exportar las rutas obtenidas a un ordenador en forma de ficheros GPX. De esta forma la latitud, longitud y el tiempo fueron obtenidos para cada posición. El fichero GPX fue analizado usando gpxpy (GPX file parser)[2]. Todos los cálculos fueron desarrollados en Phyton[3]. Excepto la conversión de coordenadas geográficas a UTM (Universal Transverse Mercator) que fue desarrollado en Java utilizando el datum WGS84 (World Geodetic System 1984). Todas las gráficas mostradas en esta memoria se obtuvieron utilizando la librería de Python matplotlib [4]. Las coordenadas de las posiciones estáticas analizadas fueron obtenidas de mapas de alta precisión proporcionados por UCS.

2.3. Análisis de posiciones estáticas

Esta sección analiza la precisión y la repercusión sobre esta de varios parámetros que los sistemas de localización en iOS proporcionan para posiciones estáticas. Además, mide el tiempo para la primera posición y compara la precisión del iPhone con la del reloj Garmin.

2.3.1. Comparación del error horizontal para diferentes precisiones deseadas

Los servicios de localización en iOS permiten al desarrollador configurar la precisión deseada para la localización en metros. Esta sección analiza la influencia de este parámetro, el cual se configura mediante la propiedad *desiredAccuracy*, sobre la precisión real. Los valores analizados en este estudio son los que proporciona el sistema operativo por defecto y pueden ser observados en la tabla 2.1.

[2] `https://github.com/tkrajina/gpxpy`
[3] `http://www.python.org`
[4] `http://matplotlib.sourceforge.net`

Tabla 2.1: Valores de la propiedad desiredAccuracy utilizados en este análisis

Name of the accuracy constant	*value*
kCLLocationAccuracyBestForNavigation	-2
kCLLocationAccuracyBest	-1
kCLLocationAccuracyNearestTenMeter	10
(custom value)	50

En la figura 2.2 se puede observar el error horizontal en metros para las diferentes precisiones deseadas. Como era de esperar, se puede observar que los resultados son muy parecidos para las precisiones *best for navigation, best* and *nearest 10 m*, mientras que son algo más imprecisos para *nearest 50 m*. Aunque la mayoría de las posiciones tienden a estar en la diagonal que van del primer al tercer cuadrante, no se puede sacar ninguna conclusión de ello debido a las limitaciones del experimento. La tabla 2.2 muestra los números exactos de este error. Como puede comprobarse la diferencias entre utiliza los tres parámetros más precisos son mínimas. Incluso siendo algo mejores para *nearest 10m*. Apple recomienda en la documentación oficial solo utilizar best for navigation en el caso de que el dispositivo este conectado a una fuente de energía por lo que tanto *best* como *nearest 10m* son buenos valores a utilizar en el sistema que se esta desarrollando en este proyecto.

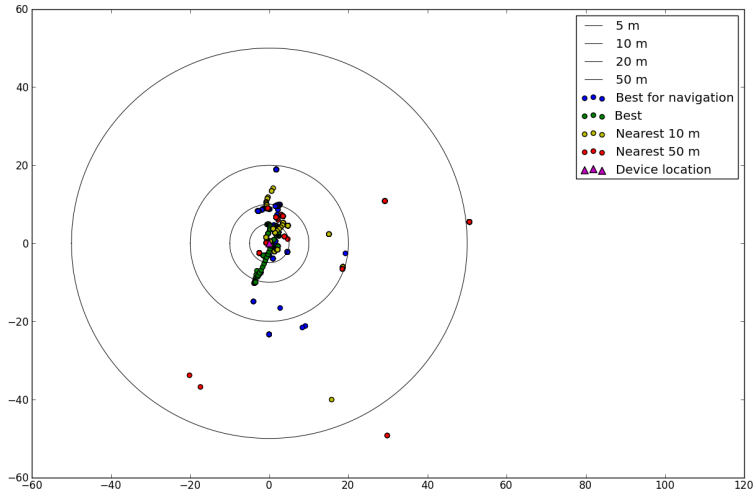


Figura 2.2: Gráfica de las distintas posiciones recibidas para diferentes precisiones deseadas

Tabla 2.2: Error horizontal en metros para las diferentes precisiones deseadas analizadas

Accuracy	*min*	*max*	*mean*	*RMSE*
Best for navigation	1.3581	23.3408	9.1048	11.2679
Best	0.5834	19.5479	9.2488	11.6861
Nearest 10 m	0.4516	43.0338	8.7107	11.3551
Nearest 50 m	0.7861	57.5612	17.2975	26.0383

Zandbergen desarrollo un estudio similar utilizando un iPhone 3G[Zan09], obteniendo un error medio de 6.9 m, con un error mínimo de 0.4 m máximo de 18.5 m. En nuestro caso el error medio es 2 metros mayor, esta diferencia se puede atribuir principalmente a que en el estudio de Zandbergen el dispositivo estaba colocado en una posición ideal mientras que en nuestro caso estaba colocado en el bolsillo del pantalón.

2.3.2. Precisión observada versus estimada

Los mismos datos experimentales que en la sección anterior fueron utilizados en esta sección para comprobar como de preciso es el parámetro *horizontalAccuracy* que iOS proporciona. Este parámetro representa una aproximación del error en metros que esa posición tiene. Durante este análisis se obtuvieron posiciones cuyo error variaba de 15 m a 149000 m. Los valores más comunes son por debajo de 100 m.

La figura 2.3 muestra el error real, el cual se ha calculado, frente al error estimado por iOS. Las conclusiones que se pueden sacar de esta gráfica son que el error predicho es siempre mayor al real. Aunque la correlación entre el predicho y el real es muy baja, existe como se puedo observar en el capítulo 2.4.3 en el apéndice D. Para lo que realmente es útil este parámetro es para diferenciar las posiciones muy imprecisas de las que no lo son tanto ya que estas suelen tener valores de este mayores de 100m.

2.3.3. Tiempo para la primera posición (time to first-fix)

A parte de la precisión de las posiciones recibidas, el tiempo que se tarda en obtener la primera posición es un parámetro también muy importante. Esto se mide normalmente con el llamado en ingles time to first fix (TTFF). En este estudio solo el inicio frío (cold start en inglés) es considerado, el cual mide el

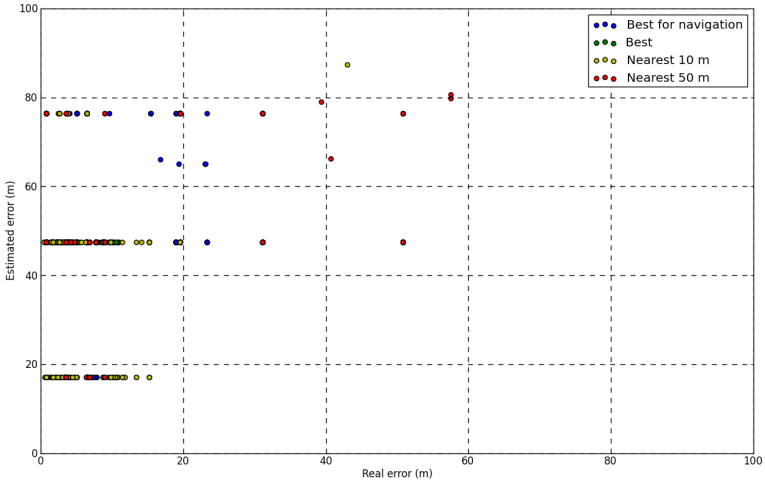


Figura 2.3: Comparación entre el error horizontal estimado por iOS y el real

tiempo en recibir la primera posición cuando el dispositivo no tiene ninguna información que ayuda a la localización almacenada en la memoria. Puesto que Apple no proporcionan ninguna información de cómo los sistemas de localización almacenan la información.

Hay que considerar que los resultados de este estudio dependen enormemente de la velocidad de la conexión a internet disponible durante el test. Ya que gran parte de la información que los sistemas de localización utilizan es descargada de internet. El estudio fue llevado a cabo para tres áreas, en cada una de ellas se calculó la posición en interior y exterior. Este estudio considera como primera posición la primera que tiene un error horizontal igual o mejor de 100 m. Como puede observarse en los resultados de la tabla 2.3, en todos los casos se recibe una posición suficientemente precisa en los primero 15 segundos y en la mayor parte de los casos en menos de 10. Por lo que podemos concluir que la velocidad y la disponibilidad de los sistemas de localización en iOS son dos de sus mejores características.

2.3.4. Comparación entre diferentes dispositivos

Para terminar con el análisis de las posiciones estáticas, el iPhone 3GS el cual ha sido utilizado para todo el análisis es comparado con el Garmin forerunner

Tabla 2.3: Tiempo en segundos para recibir una posición con un error horizontal estimado menor que 100 m

Location	*Time (s) indoor*	*Time (s) outdoor*
308	5.45	6.58
321	9.74	3.27
404	4.81	13.21
mean	6.67	7.69

405CZ. Como puede observarse en la figura 2.4, las posiciones obtenidas con el Garmin son en casi todos los casos más precisas que las del iPhone. Un hecho a remarcar es que todas las posiciones obtenidas por el reloj Garmin se encuentran dentro del círculo de radio 10m. Esto significa que cuando el Garmin proporciona una posición esta tiene siempre una precisión de por lo menos 10 m.

Como puede observarse en la tabla 2.4, las posiciones que el iPhone calcula son en media 1.7 metros más imprecisas. Además el error medio en este análisis para el iPhone es 1 metro menor, por lo que se puede concluir que el Garmin es en media por lo menos 2 metros más preciso que el iPhone.

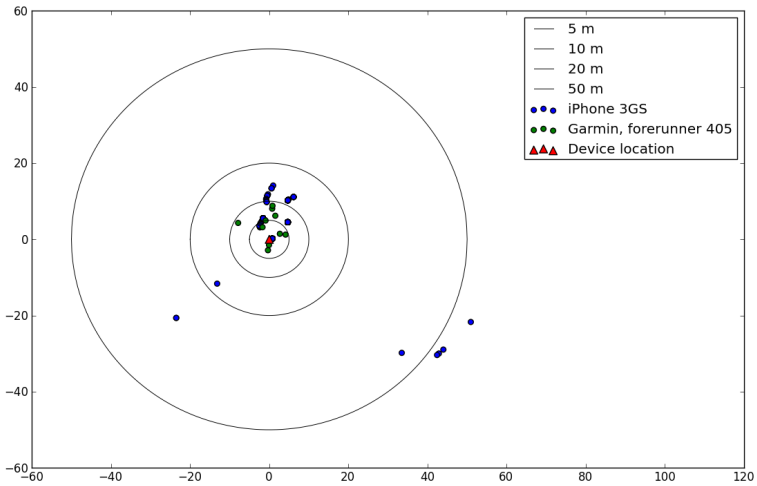


Figura 2.4: Gráfica de las distintas posiciones recibidas por iPhone 3GS y Garmin forerunner 405 en metros

Tabla 2.4: Comparación del error horizontal entre un iPhone 3GS y un Garmin forerunner 405 en metros

Device	*min*	*max*	*mean*	*RMSE*
iPhone	0.7970	55.3629	6.9375	8.8528
Gramin watch	1.4947	8.9773	5.263	5.8425

2.4. Análisis de posiciones en movimiento

Esta sección analiza la precisión que el iPhone 3GS tiene en posiciones en movimiento. La precisión real, la estimada, el tiempo entre posiciones recibidas y entre posiciones diferentes recibidas fueron analizados para encontrar patrones repetitivos para diferentes acciones que pueden suceder en una ruta como que el usuario entre en un edificio.

2.4.1. Como medir la ruta real

En el primer problema que surge cuando se quiere medir el error en una ruta es como se puede medir la ruta real. Ya que para medir el error real hay que tener en cuenta no solo las posiciones por las que paso el usuario sino también en que momento paso por cada posición.

Una primera solución podría ser dejar el tiempo fuera del análisis como se realizo en el estudio [ZB11]. Pero en una aplicación de seguimiento como la que se esta desarrollando en este proyecto el tiempo es un parámetro clave por lo que tiene que ser tenido en cuenta en el análisis. Además en este análisis se observo que iOS tiende a proporcionar la misma posición varias veces hasta que proporciona una nueva, cual puede indicar que el tiempo proporcionado en estas posiciones no es muy preciso.

Así pues se necesita medir las posiciones de la ruta real y el instante temporal en el que se paso por ellas. Debido a los medios de los que se dispone en este análisis la forma más sencilla de hacer esto era utilizando otro receptor GPS más preciso que el iPhone. El Garmin forerunner 405 CX fue utilizado para esta función. Este dispositivo es más de 2 metros más preciso que el iPhone en posiciones estáticas como observó en la sección anterior pero mucho más preciso en posiciones en movimiento como se puede observar en el apéndice A. Proporciona nuevas posiciones en menos tiempo y estas son más precisas que las proporcionadas por el iPhone.

Se calculo el error para cada posición que el iPhone proporciono, asumiendo que la velocidad entre dos posiciones consecutivas del Garmin era constante. Así pues el instante en el que el iPhone recibió la posición es calculado para obtener un punto en la ruta del garmin. La distancia entre estos dos puntos se calcula como la distancia euclídea de sus coordenadas UTM. Para representar las rutas en un mapa se utilizo Google static maps API [5]. Como puede verse en el apéndice A, 9 rutas fueron medidas durante los meses de febrero y marzo del 2012 para el desarrollo de este proyecto.

2.4.2. Error horizontal calculado

Analizando el error medio de los tres diferentes tipos de rutas que son presentados en la tabla 2.6, puede observarse que tienen diferentes valores. Éste es mayor para las rutas que cruzan un edificio que par el resto. Este resultado es el esperado ya que el dispositivo recibe las señales procedentes de los satélites GPS mucho más atenuadas cuando entra dentro de un edificio, aumentándose el error de las posiciones recibidas. Las rutas rectas deberían tener mejor error medio que las rutas con curvas pero como puede observarse esto solo se cumple para una de las rutas rectas. La ruta "From Nils Koppels Alle to Knuth-Wintherfeldts biking" tiene un error medio de 30.32 lo que la hace la ruta con más error medio entre todas las exteriores analizadas. Analizando esta ruta en el apéndice A, se puede observar que esta ruta es la única que recibe una nueva posición cada segundo. Así que se puede asumir que el tiempo que iOS proporciona para cada posición a esta frecuencia no es el mas preciso.

Tabla 2.5: Error horizontal para las diferentes rutas analizadas

Route	*Min*	*Max*	*Mean*	*RMSE*
Rutas que entran dentro de un edificio				
From 308 to 342	7.46	188.34	26.59	38.80
From 341 to 101	2.28	175.08	30.86	40.05
From 343 to 308	1.04	133.87	36.23	44.07
From library to 229	6.13	132.91	35.49	47.02
Rutas exteriores con curvas				
From 115 to library	4.24	99.34	24.94	30.97
From 229 to 115	3.24	61.52	26.33	29.92
From Nordvej to 302 biking	4.62	93.08	26.57	33.60
Rutas exteriores rectas				
From Knuth-Wintherfeldts to Nils Koppels Alle walking	3.76	41.47	17.83	19.02
From Nils Koppels Alle to Knuth-Wintherfeldts biking	4.69	218.90	30.32	38.33
All	1.04	218.90	27.13	35.10

[5]Static Maps API V2 Developer Guide `https://developers.google.com/maps/documentation/staticmaps/`

Como puede observarse el error medio para todas las rutas es de 27.3 m, el cual es bastante mayor que el obtenido en posiciones estáticas. Esto se debe a dos razones, a que realmente el error es mayor y a que el método utilizado para calcularlo introduce un error adicional. Como puede verse en el histograma en la figura 2.5, la mayor parte de las posiciones tienen un error comprendido entre 10 y 20 m.

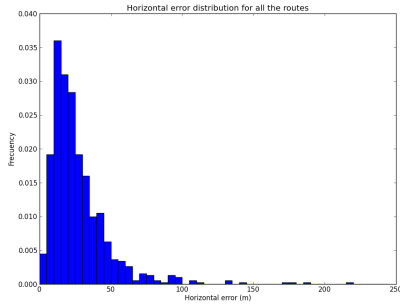


Figura 2.5: Histograma del error horizontal para todas las rutas

2.4.3. Filtrado de las posiciones más imprecisas

Aunque Apple no proporciona ninguna información de cómo diferenciar entre que sistema de posicionamiento ha sido utilizado en cada posición. En este estudio se ha diseñado un sistema basado en la observación de las posiciones recibidas. Si la posición ha sido obtenida utilizando posicionamiento Wi-Fi este valor es de 65 o 100 mientras que si se ha obtenido utilizando posicionamiento celular es un numero sin decimales y normalmente mayor que 1.000. Mientras que en el caso de que la posición ha sido obtenido utilizando GPS este parámetro tiene siempre posiciones decimales. Este descubrimiento va a ser muy útil a la hora de decidir que posiciones deben de ser filtradas.

En todos los mapas de esta memoria las líneas azules indican las rutas obtenidas con el Garmin, las rojas son las obtenidas por el iPhone y las verdes son las obtenidas después de aplicar un filtro a las rutas de este último. Como se puede observar en los mapas del apéndice A en la rutas rojas hay partes en las que la ruta es bastante precisa pero hay otras en la que no lo es. Dos ejemplos de estas se pueden observar en la figura 2.6. Estas posiciones deben ser filtradas para obtener un seguimiento de los usuarios mas preciso ya que no proporcionan ninguna información relevante y hacen el seguimiento mucho menos preciso.

Analizando las gráficas de error que se encuentra en el apéndice A, se descubrió

Figura 2.6: Filtro WiFi. Dos ejemplos de posiciones muy imprecisas (dentro de los círculos amarillos) en las rutas: from 343 to 308 (izquierda) and from Nils Koppels Alle to Knuth-Wintherfeldts (derecha)

Tabla 2.6: Error horizontal para diferentes filtros de error y Wi-Fi en metros

Error filter	*Wi-Fi filter*	*Min*	*Max*	*Mean*	*RMSE*
1000	0	1.04	218.90	27.13	35.10
200	0	1.04	105.53	24.85	29.67
150	0	1.04	105.53	24.35	29.09
100	0	1.04	105.53	24.21	28.91
100	2	1.04	105.53	24.19	28.90
100	3	1.04	105.53	24.19	28.90
100	4	1.04	105.53	23.91	28.38
100	5	1.04	105.53	23.91	28.38
100	6	1.04	105.53	23.99	28.46
100	7	2.35	105.53	23.98	28.48

que todas estas posiciones tenían un error horizontal estimado muy elevado. En la mayor parte de los casos era mayor de 700m y siempre mayor que 100m. estas posiciones son obtenidas utilizando GPS con mala calidad de señal o posicionamiento celular.

Como puede verse en la tabla 2.6, simplemente filtrando las posiciones con error estimado mayor de 100 m el error medio desciende de 27.1 m a 24.2 m. Pero, analizando los datos se observo que posiciones como las que se muestran en la parte izquierda de la figura 2.7 eran muy imprecisas aunque su error estimado era menor que 100m. Se descubrió que estas posiciones se obtenían mediante posicionamiento Wi-Fi. Pero no todas estas posiciones tienen que ser eliminadas ya que cuando el usuario se encuentra dentro de un edificio estas son

Figura 2.7: Dos ejemplos en los que se ha eliminado posiciones obtenidas utilizando posicionamiento Wi-FI (dentro de los círculos amarillos) en las rutas: from 341 to 101 (arriba) and from 343 to 308 (abajo)

las posiciones más precisas que se obtienen. Así pues, el filtro Wi-Fi diseñado solo deja pasar las posiciones Wi-Fi que aparecen seguidas en grupos de tamaño igual o menor que el indicado. Este filtro fue diseñado para ejecutarse después del filtro de error horizontal estimado.

Como se puede observar en la tabla 2.6, los mejores resultados se obtienen para filtro de error de 100 m y filtro Wi-Fi de 4 o 5. El error medio a sido mejorado casi 4 m, simplemente ejecutando filtros simples.

Tabla 2.7: Tiempo entre diferente posiciones en segundos

Route Name	*Min*	*Max*	*Mean*	*RMSE*
From 308 to 342	0.06	72.56	16.23	26.59
From 341 to 101	0.50	104.11	7.89	19.25
From 343 to 308	0.99	140.00	23.90	39.77
From library to 229	1.62	104.85	32.83	45.96
From 115 to library	7.00	64.00	23.44	29.34
From 229 to 115	1.58	61.96	29.83	34.06
From Nordvej to 302 biking	0.99	30.00	4.61	7.19
From Knuth-Wintherfeldts to Nils Koppels Alle walking	0.93	51.65	3.47	7.47
From Nils Koppels Alle to Knuth-Wintherfeldts biking	0.97	38.00	1.63	3.79
Total	0.06	140.00	6.50	15.80

2.4.4. Tiempo y velocidad entre diferentes posiciones

En esta sección las rutas utilizadas son filtradas con filtro de error de 100 m y Wi-Fi de 4 muestras. Además solo las posiciones únicas son tenidas en cuenta. Saber el tiempo entre nuevas posiciones es importante para diseñar los algoritmos de seguimiento en el modelo. Ya que se deben diseñar de forma que puedan resolver caminos entre posiciones separadas esa distancia.

Como puede observarse en la tabla 2.7, el tiempo medio entre nuevas posiciones para todas las rutas es de 6.5 segundos. Pero el tiempo máximo en recibir una posición nueva es de 140 segundos. Los algoritmos de representación en el modelo 3D deberán diseñarse teniendo en cuenta estos tiempos.

También se realizó un estudio de la velocidad entre las distintas posiciones el cual se puede encontrar en la sección 2.5.5 en el apéndice D. El cual concluye que la velocidad entre dos puntos consecutivos varía bastante, debido a que el instante temporal proporcionado para cada posición no es nada preciso. Así que la forma más precisa de calcular la velocidad es calculando la velocidad media de toda la ruta.

2.4.5. Detectar cuando el usuario entra a un edificio

Para representar a los usuarios en el modelo 3D de la forma más realista posible, sería muy útil poder detectar cuando este entra en un edificio. Después de aplicar el filtro diseñado en el apartado anterior, solo quedan posiciones obtenidas utilizando posicionamiento Wi-Fi cuando el usuario se encuentra en el

interior de un edificio. El resto de posiciones Wi-Fi son eliminadas al pasar por el filtro. Así pues detectar cuando un usuario entró a un edificio es tan fácil como detectar las posiciones obtenidas por posicionamiento Wi-Fi que pasan el filtro, es decir las que tienen error horizontal estimado igual a 65m. Como puede observarse en la figura 2.8, esta teoría se cumple en todos los casos analizados en este estudio.

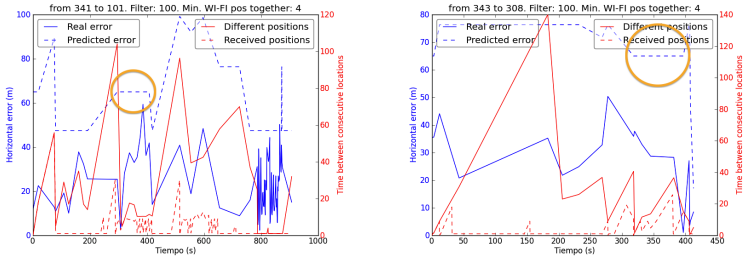


Figura 2.8: Dos gráficas en la que se detecta que el usuario estaba dentro de un edificio (dentro de los círculos amarillos)

Capítulo 3

Representación de las rutas en el modelo 3D

Este capítulo explica porque los datos que los sistemas de localización ofrecen no pueden ser utilizados directamente para posicionar a los usuarios en el modelo. Además detalla los algoritmos que han sido diseñados para resolver este problema y los resultados obtenidos.

3.1. Descripción del problema

Como fue estudiado en el capítulo 2, las posiciones que son recibidas del iPhone tienen un error medio de casi 10 m en el mejor de los casos. El modelo 3D que se utiliza para la representación de los usuarios tiene una precisión del orden de centímetros. Además Realsite utiliza tecnología de videojuegos para reproducir estos modelos 3D. Es decir que los avatares de los usuarios se van a mover por un mundo virtual el cual tiene las mismas reglas que el mundo real, hay gravedad, no se pueden atravesar muros, no se pueden escalar superficies a partir de una cierta inclinación... Además lo avatares tiene que parecer reales en el modelo, las rutas que sigan tendrán que ser realistas. Por estas razones hay que diseñar unos algoritmos que tomen las posiciones que el iPhone obtiene y devuelvan una ruta libre de obstáculos. Los problemas que tienen que ser resueltos son

presentados en las siguiente secciones.

3.1.1. Transformación de coordenadas

El primer problema que surge al intentar representar los usuarios en el modelo es que las posiciones son obtenidas en coordenadas geográficas, pero el modelo esta representado usando un sistema coordenado propio. Así pues hay que desarrollar un programa que realice esta transformación de coordenadas.

3.1.2. Atravesando edificios y posiciones encima de los edificios

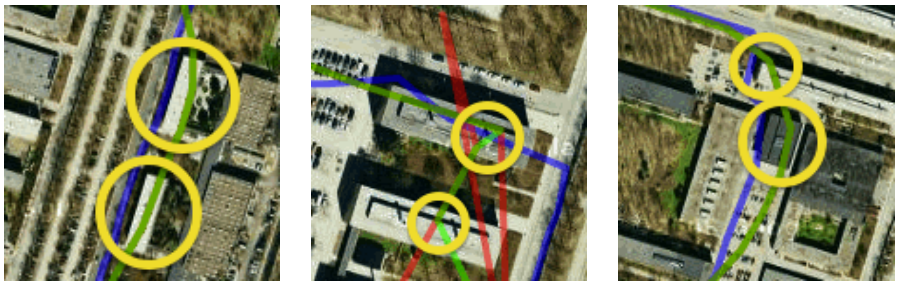


Figura 3.1: Diferentes ejemplos de rutas atravesando edificios y posiciones en los tejados de los edificios (dentro de los círculos amarillos)

El problema de rutas atravesando edificios surge cuando el camino que une dos posiciones atraviesa un edificio, este problema es muy popular ya que el campus tiene muchos edificios. Este problema tiene que ser solucionado porque el avatar choca contra el edificio y algunas ocasiones se queda atascado. Otro problema que pasa normalmente es recibir posiciones que en lugar de encontrarse en el suelo se encuentran en los tejados de los edificios. Estas posiciones son inalcanzables para un usuario real en el campus y tienen que serlo también para los avatares. Por lo que estas posiciones tienen que ser borradas o movidas al suelo. Ambos problemas pueden observarse en la figura 3.1.

3.1.3. Cambios de altitud en la ruta

El campus de la DTU no esta todo a la misma altura. Tiene varios muros de varios metros de altitud que no pueden ser atravesados directamente. Igual que

pasaba en el caso de los edificios el avatar puede chocarse y quedarse enganchado en el muro. Ejemplos de este problema se pueden ver en la figura 3.2.

Figura 3.2: Diferentes ejemplos de rutas en las que hay un cambio de altitud (dentro de los círculos amarillos), debido a que hay un muro

3.1.4. Cambio rápido de dirección

Como se descubrió en la sección 2.4.4, en algunas circunstancias se obtiene una posición nueva cada segundo. Estas posiciones están muy próximas unas de otras y no suelen estar alienadas, lo que hace que el avatar cambie de dirección cada muy poco tiempo. Esto quita sensación de realismo.

3.2. Herramientas y conceptos relacionados

El problema que se tiene que resolver para representar a los usuarios en el modelo esta relacionado con técnicas de path finding y de map matching. Path finding es una técnica utilizada mayormente en videojuegos, la cual estudia el proceso de encontrar la ruta más corta entre dos puntos en un mapa 3D digital. Estas técnicas necesitan el modelo representado como una red compuesta de nodos y conexiones como puede verse en [PEH68], [YB06], [Mul04] and [Rab00]. Un ejemplo de un sistema parecido al desarrollado en este proyecto, el cual desarrolla un juego Pac-Man , puede encontrarse en [MYW10]. Y otras aplicaciones prácticas en [MCP], [SB00]. Map matching es el proceso de correlacionar una secuencia de posiciones de usuarios con una red de posibles caminos. Diferentes estudios de esta tecnología en diferentes situaciones se pueden encontrar en [CEW00], [GRJZ04], [FM04], [Zho], [TG11], [YJs05].

Estos algoritmos para representar a los usuarios en el modelo tendrán que ejecutarse en el reproductor del modelo 3D, el cual se ejecuta en cualquier navegador

de escritorio en realsite.dk. Este ha sido desarrollado utilizando Unity 3D[1], el cual se utiliza en el desarrollo de videojuegos y contenido 3D interactivo. Tiene unas herramientas de path finding que podrían haber sido de mucha utilidad, pero necesitan que el terreno sea un objeto estático y en realsite es un objeto dinámico. Por lo que no se podía utilizar estar herramientas. Así pues todos los algoritmos para posicionar a los usuarios en el modelo tenían que ser desarrollados utilizando métodos disponibles en la librería physics[2]. Esta librería posee métodos simples de interacción de objetos como por ejemplo el métdo RayCast[3].

3.3. Solución diseñada

3.3.1. Requerimientos del sistema

Como ya ha sido comentado en la sección anterior, es necesario diseñar unos algoritmos que representen a los usuarios en el modelo correctamente. Los requerimientos de este sistema son una mezcla de los que UCS impuso y los resultados del estudio del capítulo 2:

- El usuario se debe de poder mover por las nuevas posiciones sin quedar atascado y parecer real.
- El sistema tiene que trabajar en realsite no solo con el modelo 3D del campus de la DTU pero también con otros modelos.
- La ruta obtenida tiene que estar lo más cerca posible de la real.
- Todos los algoritmos tiene que desarrollarse en Unity usando el lenguaje de programación C#.

Los algoritmos desarrollados en este proyecto cumplen todos estos requerimientos.

[1] `http://unity3d.com/`
[2] `http://unity3d.com/support/documentation/ScriptReference/Physics.html`
[3] `http://unity3d.com/support/documentation/ScriptReference/Physics.Raycast.html`

3.3.2. Transformación de coordenadas geográficas a DTU y viceversa

La conversión de coordenadas se realiza en el servidor en lugar de en el reproductor de realsite, por lo que tuvo que ser desarrollada en jaca. El modelo 3D de la Technical University of Denmark usa como sistema de referencia un sistema coordenado cuyo punto 0,0 está en el centro del campus. UCS disponía de un documento procedente de la empresa que tomo las medidas del campus, el cual puede verse en el apéndice B. Las fórmulas que se calcularon para realizar el cambio de coordenadas de DTU a UTM pueden verse a continuación:

$$\begin{pmatrix} DTU_x \\ DTU_y \end{pmatrix} = \lambda \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} UTM_x \\ UTM_y \end{pmatrix} + \begin{pmatrix} \Delta x \\ -\Delta y \end{pmatrix}$$

$$\begin{pmatrix} UTM_x \\ UTM_y \end{pmatrix} = \frac{1}{\lambda} \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \frac{\sin(\theta)-\sin(\theta)^3}{\cos(\theta)\sin(\theta)} \end{pmatrix} \begin{pmatrix} DTU_x - \Delta x \\ DTU_y + \Delta y \end{pmatrix}$$

$$\lambda = 0{,}9998133$$

$$\theta = 14{,}3472^\circ$$

$$\Delta x = 834864{,}7242936$$

$$\Delta y = 6172294{,}56110385$$

Ya que iOS y la mayoría de dispositivos que miden la posición utilizan coordenadas geográficas, hay que realizar la conversión de estas coordenadas a coordenadas UTM. Esta conversión no es directa, ya que hay muchas soluciones gratuitas en internet para este problema algunas de ellas pueden encontrarse en [4] [5] [6]. Geographic/UTM Coordinate Converter fue la solución seleccionada, tuvo que ser traducida de JavaScript a Java y modificada para que Copenhague estuviera en la zona 32U en lugar de en la 33U que es en la que en realidad se encuentra, ya que esta zona era la utilizada en el sistema de referencia.

[4] Coordinate conversions made easy `https://www.ibm.com/developerworks/java/library/j-coordconvert/`

[5] Geographic/UTM Coordinate Converter `http://home.hiwaay.net/~taylorc/toolbox/geography/geoutm.html`

[6] Convert Between Geographic and UTM Coordinates `http://www.uwgb.edu/dutchs/usefuldata/ConvertUTMNoOZ.HTM`

3.3.3. Algoritmos diseñados para representar el avatar en el modelo

El sistema diseñado fue dividido en cinco algoritmos diferentes que son ejecutados uno después de otro. Cada uno de estos algoritmos soluciona alguno de los problemas de los expuestos en el capítulo anterior y no estropea lo solucionado por los algoritmos que han sido ejecutados anteriormente. Los algoritmos son ejecutados en el mismo orden que son presentados en este capítulo. En todos los mapas presentados en este capítulo la línea azul es la ruta obtenida por el Garmin, mientras que la líneas roja y verde son las obtenidas con el iPhone y tras las la transformación respectivamente. Una explicación más detallada de todos los algoritmos se puede encontrar en el capítulo 3.3.3 del apéndice D.

3.3.3.1. Paso 1: añadir las entradas de los edificios cuando el usuario entra a uno

Figura 3.3: Un ejemplo de ruta donde dos entradas de edificios son añadidas a la ruta porque el usuario estaba dentro de un edificio (dentro de los círculos amarillos)

Como fue comentado en la sección 2.4.5, después de aplicar un filtro de error de 100 m y un filtro Wi-Fi de 4 posiciones, detectar cuando el usuario está dentro de un edificio es tan fácil como detectar las posiciones Wi-Fi sobrantes. Así pues el algoritmo desarrollado en esta parte realiza las siguientes funciones:

1. Detecta las posiciones Wi-Fi en la ruta y borra las que están dentro de un edificio.

2. Añade la entrada del edifico que entre el primer punto que esta dentro de un edifico y el anterior. Realiza lo mismo para la salida pero con el último punto.

3. Añade un salto en la ruta entre las dos posiciones de las puertas.

Un ejemplo de la ruta resultante tras ejecutar este algoritmo se puede ver en la figura 3.3.

Este algoritmo es ejecutado en primer lugar porque las posiciones Wi-Fi no pueden ser filtradas antes de ejecutarlo y además puede estropear alguno de los problemas que los algoritmos posteriores solucionan. Las nuevas rutas no contienen ninguna información de donde se situaba el usuario cuando estaba dentro del edificio, pero esto no es una pérdida de información ya que estas posiciones son generalmente muy poco precisas.

3.3.3.2. Paso 2: filtrado de las rutas usando el algoritmo de Douglas-Peucker

Como fue explicado en la sección 3.1.4, el problema cambio rápido de dirección se producía cuando las posiciones no estaban muy separadas unas de otras. Así pues algún algoritmo de simplificación de curvas tenía que ser desarrollado, ejemplos de estos algoritmos pueden verse en [DD73], [BKR92], [Ram72]. Entre ellos el que proporcionaba mejores resultados en este caso y por lo tanto el algoritmo implementado fue el algoritmo de Doublas-Peucker [DD73]. Este algoritmo es un algoritmo recursivo que reduce el número de puntos necesario para representar una curva manteniendo la precisión. Si a la ruta se le han añadido puertas de edificios, la ruta es dividida por los saltos antes de ejecutar este algoritmo de forma que las posiciones de las puertas no son filtradas.

Un ejemplo de ejecución de este algoritmo se puede ver en la figura 3.4. En nuestro caso un valor de ϵ de 5 m solucionaba completamente el problema.

3.3.3.3. Paso 3: ajuste de la ruta con la red de calles

Como se comentó en la sección 3.2, todos los algoritmos de map matching necesitan una red de carreteras con la que trabajar. Ya que no se podía obtener una del modelo directamente, un fichero de AutoCAD, que fue usado en el desarrollo del modelo, fue utilizado para obtener esta red. Esta red contiene las principales

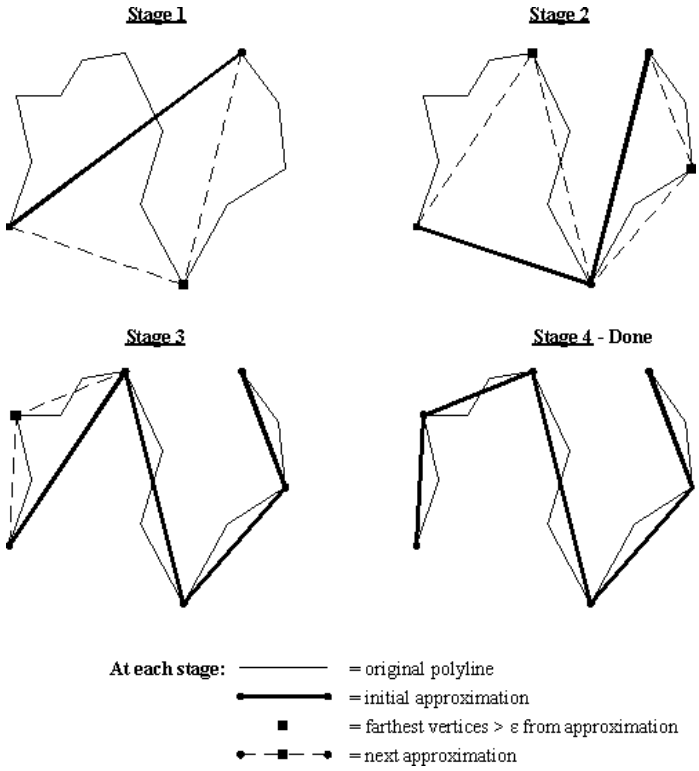


Figura 3.4: Algoritmo de Douglas-Peucker[a]

[a] `http://softsurfer.com/Archive/algorithm_0205/Pic_DP-2.gif`

calles del campus en forma de segmentos. Por lo que no todas las posiciones pueden ser proyectadas en esta red solo las que realmente pasaron por esas calles, ya que la red de calles real es mucho más grande.

El algoritmo de map matching desarrollado es una variante del algoritmo 2 presentado en [CEW00]. El algoritmo proyecta una posición en la red de calles si la proyección esta más cerca que un umbral y si la diferencia entre la orientación de ambas es menor que 30 grados. El umbral de distancia es 20 o 30 m dependiendo si el punto anterior ha sido proyectado en ese segmento o no. La proyección es calculada como la intersección entre el segmento perpendicular al segmento de la red que pasa por el punto analizado. Un ejemplo de la ejecución de este algoritmo puede verse en la figura 3.5.

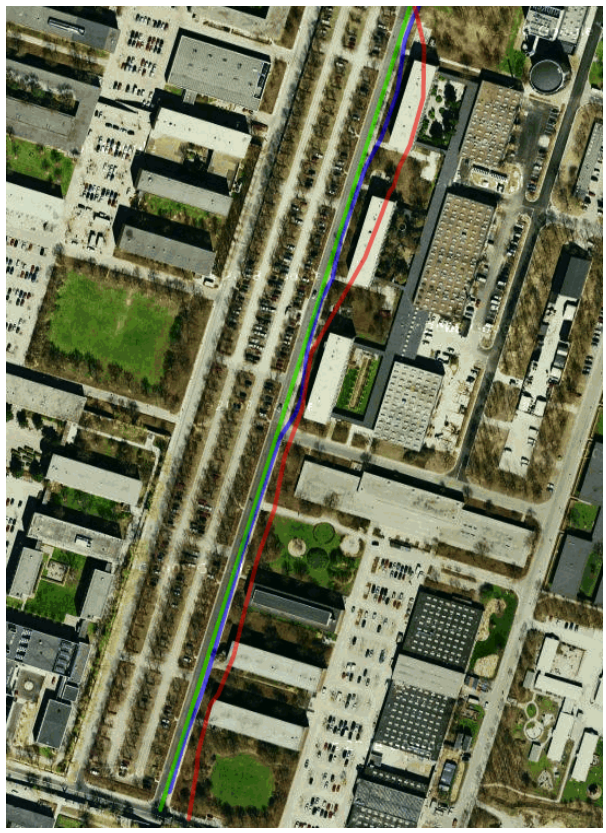


Figura 3.5: Ejemplo de rutas mejoradas usando el algoritmo de map matching

3.3.3.4. Paso 4: corrección de las diferencias de altitud

Como fue comentado en la sección 3.1.3, el campus de la DTU no es completamente plano, hay varios lugares en los que hay muros que el carácter no puede escalar. Para pasar estos muros el usuario puede utilizar escaleras que se encuentran en el exterior o en el interior de edificios. Nuestro algoritmo considera los dos casos.

El algoritmo detecta si dos puntos consecutivos están a una diferencia de altitud mayor de 4 m. Si lo están añade una serie de puntos que hacen que el carácter suba por las escaleras o entre y salga de un edificio. En el caso de las escaleras exteriores un fichero con las posiciones que tienen que ser añadidas fue creado manualmente ya que las escaleras están incluidas en el mismo objeto que el terreno en el modelo. Esto hacía imposible detectarlas usando los métodos de

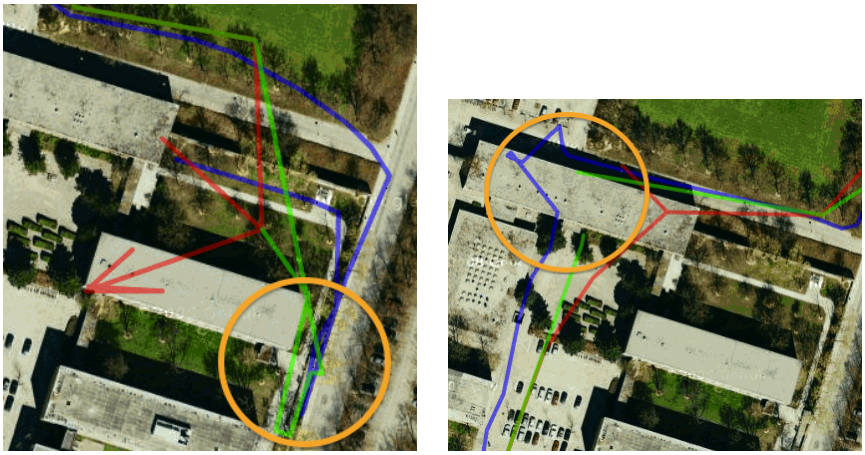


Figura 3.6: Ejemplo de como arreglar las diferencias de altitud en dos rutas añadiendo escaleras exteriores (izquierda) o puertas (derecha)

Unity. Para las entradas de los edificios se detectaba si había algún edificio cerca del cambio de altitud y si este tenía una puerta en una altura y la otra en la otra. Un ejemplo de la ejecución de este algoritmo se puede ver en la figura 3.6.

3.3.3.5. Paso 5: evitar obstáculos conocidos (edificios) en la trayectoria

El objetivo de este algoritmo es generar la ruta final libre de obstáculos, la cual pueda ser seguida por un carácter en el modelo sin chocar con ningún obstáculo. Este problema fue expuesto en la sección 3.1.2. Este algoritmo hace dos tareas fundamentalmente, añade puntos cerca de las esquinas de los edificios para rodearlos y elimina todas las posiciones que están encima de edificios. Pero estos últimos no son borrados directamente, son utilizados antes de ser borrados. Los puntos que son añadidos están 2 m en el eje x e y lejos de las esquinas de los edificios.

El algoritmo tenia que ser capaz de añadir un número indeterminado de puntos para rodear los edificios, además los edificios suelen estar unidos unos con otros por lo que tiene que ser capaz de resolver edificios con formas complejas y no solo con forma rectangular. Para cumplir estos criterios el algoritmo diseñado es un algoritmo recursivo.

El algoritmo comprueba si hay algún obstáculo en la línea que une dos puntos

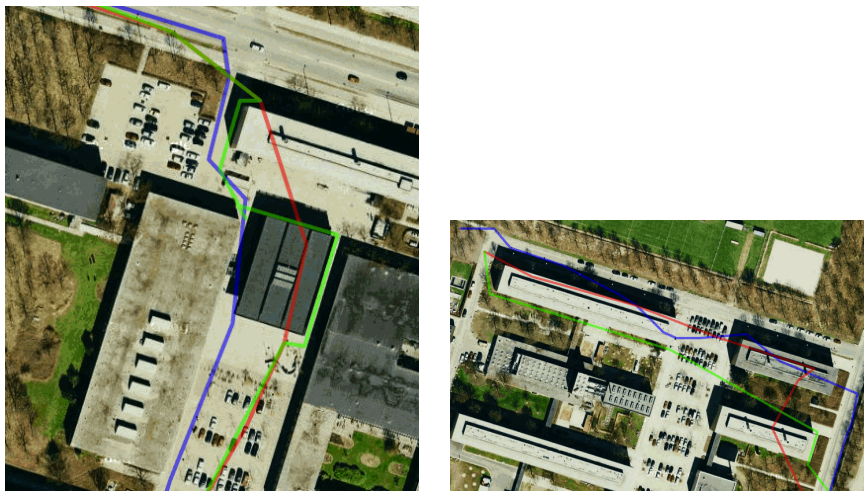
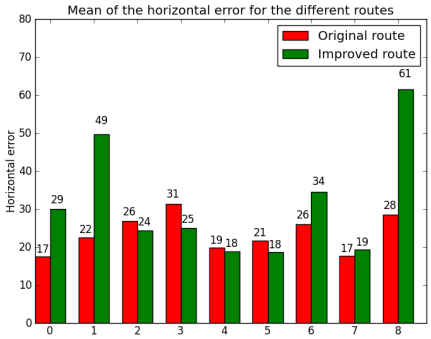


Figura 3.7: Ejemplos de ejecución del algoritmo que evita chocar con edificios

consecutivos, si lo hay comprueba si hay obstáculos entre el primer punto y alguna de las esquinas del objeto obstáculo. Si no la hay, el punto es añadido a la ruta y el algoritmo se vuelve a ejecutar recursivamente. Si hay obstáculo, se toma ese obstáculo y se vuelve a ejecutar el algoritmo que comprueba las esquinas. Si hay varias esquinas que producen caminos libres de obstáculos el que produce un camino más corto es el que se añade a la ruta. La figura 3.7 muestra dos ejemplos de la ejecución de este algoritmo.

3.4. Resultados

El sistema presentado en la sección anterior cumple todos los requerimientos que fueron presentados en la sección 3.3.1 para todas las rutas analizadas. Las rutas obtenidas junto con las originales y las de referencia pueden verse en el apéndice C. Para calcular el error de las rutas calculadas fueron contempladas varias soluciones: utilizar la misma solución que en la sección 2.4, utilizar la distancia de Hausdorff [FDA08] o utilizar la distancia de Fréchet [AM06]. La primera opción fue elegida entre otras razones para que los resultados fueran comparables con los del estudio de la sección 2.4. Para calcular el error de esta manera había que calcular el instante que se recibieron las posiciones que son añadidas a la ruta por este sistema. Este instante temporal fue calculado asumiendo velocidad constante entre las nuevas posiciones añadidas y usando los instantes temporales de las posiciones que pasaron el filtro de la ruta original.



Route name	*Number*
from 308 to 342	0
from 341 to 101	1
from 343 to 308	2
from library to 229	3
from 115 to library	4
from 229 to 115	5
from Nordvej to 302 biking	6
from Knuth-Wintherfeldts to Nils Koppels Alle walking	7
from Nils Koppels Alle to Knuth-Wintherfeldts biking	8

Figura 3.8: Comparación del error horizontal entre las rutas originales y las mejoradas con el sistema

Como puede verse en la figura 3.8, el error medio es mejorado en 4 de las 9 rutas analizadas tras ejecutar el algoritmo. Pero analizando los mapas del apéndice C se puede ver que de esas 4 rutas en las que se empeora el error medio, solo en dos casos las rutas son menos precisas que antes de ejecutar los algoritmos. Estas dos rutas son "from 308 to 342" y "from 341 to 101", durante las cuales el usuario entro dentro de varios edificios y por lo tanto los resultados que el iPhone proporcionó eran bastante imprecisos. Que el error sea mayor en rutas que son más precisas solo indica que el instante temporal de las posiciones no es el correcto, ya que como fue analizado en la sección 2.4.4 el tiempo que iOS proporciona junto con las posiciones no es muy preciso. En el resto de las siete rutas, las rutas no solo eran más precisas que las rutas antes de ejecutar el algoritmo sino que en algunas partes eran incluso más precisas que las que proporcionó el Garmin.

Capítulo 4

Sistema final

Esta sección describe la implementación final del sistema. Las diferentes partes que han sido explicadas en los capítulos anteriores, más una nueva aplicación para iOS han sido unidas para formar un sistema completamente funcional. Hay que tener en cuenta que este sistema es más una prueba de concepto que permitirá a UCS desarrollar sistemas más avanzados usando esta tecnología en el futuro.

4.1. Arquitectura del sistema

Como se puede ver en la figura 4.1, el sistema esta compuesto de tres partes principales, la aplicación para iOS que captura y envia las posiciones del usuario, el servidor con la base de datos que las almacena y la página web (realsite.dk) donde se transforman los datos y se visualizan los avatares de los usuarios en los modelos 3D. En este sistema el alumno ha desarrollado completamente la aplicación para iOS, el algoritmo para la conversión de coordenadas que es ejecutado en el servidor y el código que mejora las rutas que es ejecutado en el visualizador de las rutas de la página web. La integración de las distintas partes desarrolladas por el alumno y los servicios ya existentes de la empresa fue desarrollada por ambas partes, los trabajadores de la empresa y el alumno.

Figura 4.1: Arquitectura del sistema[a]

[a] http://cdn1.iconfinder.com/data/icons/database/PNG/512/Database_1.png http://www.jailbreakyourphone.net/wp-content/uploads/2011/07/iphone.png http://aux.iconpedia.net/uploads/1360992576.png http://upload.wikimedia.org/wikipedia/commons/thumb/c/c1/Computer-aj_aj_ashton_01.svg/2000px-Computer-aj_aj_ashton_01.svg.png

4.2. Aplicación móvil

Función La función de la aplicación móvil en el sistema es obtener, guardar, filtrar y enviar la localización de los usuarios al sistema central. El seguimiento no esta siempre activo, el usuario decide cuando activarlo pulsando un botón en la pantalla principal de la aplicación. Cada sesión de seguimiento tiene asociada una foto que es tomada utilizando la aplicación. La aplicación detecta en que modelo 3D se encuentra el usuario. Si el usuario no se encuentra en ningún lugar donde hay disponible un modelo 3D, el seguimiento no se puede iniciar. Todas las posiciones obtenidas son filtradas utilizando el filtro de error 100 m y Wi-Fi de 4 posiciones, el cual fue explicado en la sección 2.4.3, guardadas en una base de datos en el dispositivo y enviadas al servidor. Guardar las rutas en la memoria del teléfono permite al usuario explorar sesiones de seguimiento previas en un mapa utilizando la aplicación. Los parámetros de configuración que fueron usados para configurar el *CLLocationManager* son *desiredAccuracy* definido a *kCLLocationAccuracyBest* y *locationFilter* de 1 m.

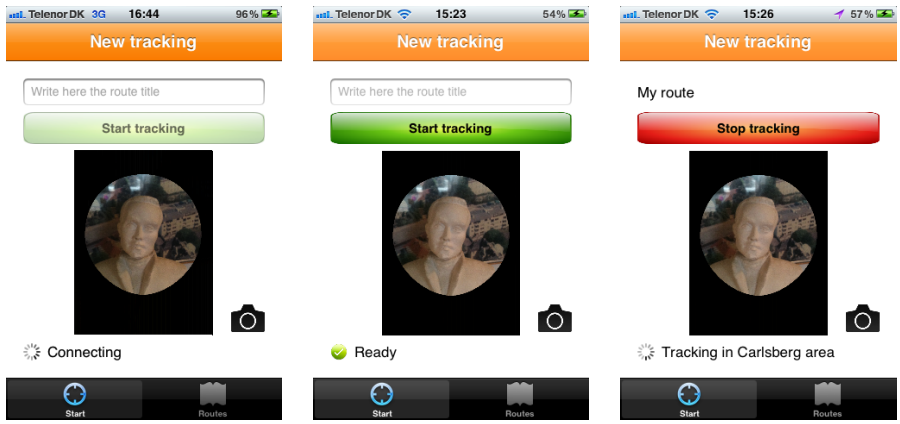


Figura 4.2: Diferentes estados de la pestaña start de la aplicación para iOS

Interfaz de usuario La interfaz de usuario de la aplicación esta compuesta de una tab bar, el cual permite cambiar entre las dos ventanas principales start and routes.

- Venta start, puede verse en la figura 4.2, muestra de arriba abajo un cuadro de texto para introducir el nombre de la sesión, un botón para iniciar o para el seguimiento, la foto que será asociada a la sesión con un botón que permite cambiarla y el estado del seguimiento. Este estado puede ser *connecting, no Internet connection, ready, starting location services, searching for GPS signals, uploading route information, tracking in x area, GPS disabled, no 3D model available in this area* y *problem with the connection to the server*, el diagrama de estados se puede ver en la figura 4.4.
- Ventana routes, figura 4.3, muestra una lista de las sesiones de seguimiento realizadas con la aplicación ordenadas por fecha. Seleccionando una de ellas se puede ver la sesión en un mapa y una ventana con cierta información.

Comunicación con el servidor La comunicación entre la aplicación y el servidor se realiza mediante peticiones HTTP GET y POST. La aplicación hace tres tipo de peticiones al servidor. La primera que se desarrolla al encender la aplicación es la de descargar la lista de áreas con modelo 3D. Tan pronto como la sesión es iniciada se manda un petición POST que contiene el título, el área y la foto asociados esa sesión, el servidor contesta con un ID para esa sesión.

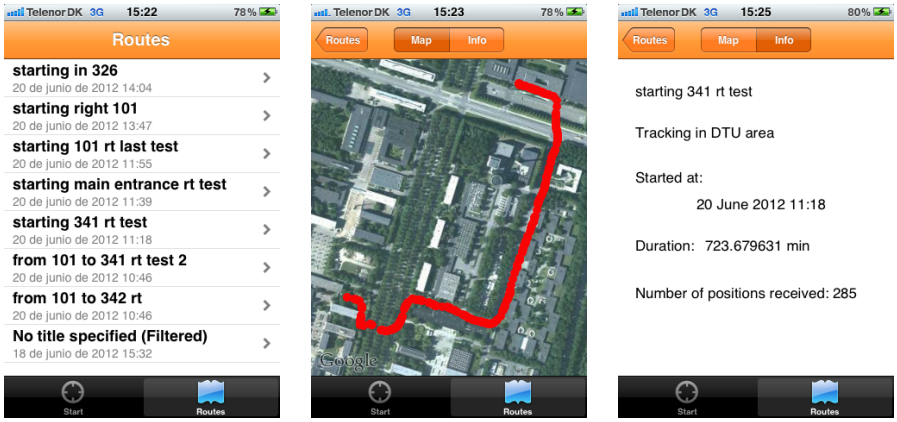


Figura 4.3: Todas las ventanas que se presentan en la pestaña route

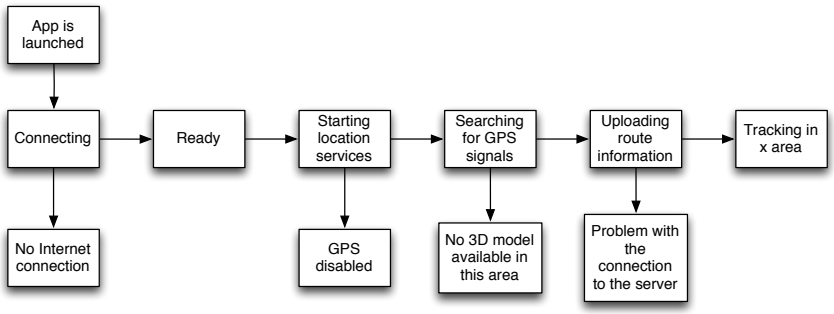


Figura 4.4: Diagrama de estados de la aplicación

Este ID es utilizada cada vez que se manda una posición al servidor utilizando una petición GET, todos los parámetros son enviados en la URL en este caso.

4.3. Aplicación en el servidor

El servidor ejecuta la aplicación web que se encarga de procesar todas las peticiones que han sido explicadas en la sección anterior y las que son necesarias para el funcionamiento de realsite.dk. Además guarda las coordenadas geográficas que la aplicación envía en una base de datos MySQL. La transformación de coordenadas geográficas a coordenadas DTU es ejecutada cuando el reproductor

del modelo 3D pide una ruta.

4.4. Reproductor en realsite.dk

El reproductor en realsite.dk es la interfaz que muestra las rutas obtenidas con la aplicación para iOS en el modelo 3D. Para visualizarlo solo hace falta un navegador de escritorio con el plugin de Unity instalado y acceder a `www.realsite.dk/DTU/public`. Una vez que el modelo 3D esta cargado hay que pulsar la tecla L para que se carguen todas las rutas disponibles en la base de datos. Estas bolas son de color rosa para su fácil identificación y tienen alrededor la fotografía que fue enviada cuando la ruta fue iniciada, la figura 4.5 muestra varias rutas en el reproductor. La transformación que fue explicada en el capítulo 3 es ejecutada también por el reproductor para cada ruta antes de que las bolas se empiecen a mover. El sistema esta hecho para funcionar en tiempo real, es decir cada posición nueva que se recibe se ejecuta el algoritmo y se añade a la ruta. La cámara en el reproductor esta siempre colocada alrededor del carácter que el usuario que esta reproduciendo el modelo puede mover. Es decir par ver las diferentes rutas hay que mover manualmente el carácter por el modelo.

Figura 4.5: Dos capturas de pantalla de realsite.dk con el sistema activado

Capítulo 5

Conclusión

Esta memoria empieza con un estudio de los sistemas de localización en iOS. Este estudio muestra que las posiciones estáticas tiene un error medio de 8 m. Comprueba que la precisión no es una de sus mejores características pero si lo son la velocidad y la disponibilidad. Una posición precisa es obtenida en los primeros 15 segundos y en los primeros 10 en la mayor parte de los casos. Además no se encontró ningún lugar en el campus en el que no se pudiera determinar la localización. No solo posiciones estáticas son analizadas, también posiciones en movimiento. El error medio en estos casos es mayor que para posiciones estáticas ya que el método utilizado para calcularlo introducía ciertos errores. Pero se presenta un filtro, el cual elimina las posiciones con error estimado mayor de 100 m y posiciones Wi-Fi que aparecen en grupos menores que 4, que mejora este error medio en casi 4 m. Después de ejecutar estos filtros la detección de cuando un usuario entra a un edificio es tan fácil como detectar las posiciones restantes obtenidas con posicionamiento Wi-Fi.

Después de este estudio se presenta un novedosos sistema para representar rutas obtenidas con un iPhone en un modelo 3D. Esta representación necesita una transformación de los datos recibidos que se lleva a cabo mediante un algoritmo dividido en cinco pasos diferentes, cada uno de los cuales resuelve un problema sin estropear lo arreglado por los algoritmos ejecutados anteriormente. Dos de estos pasos son algoritmos recursivos. Este sistema mejora el error medio en cuatro de las nueve rutas analizadas. Pero analizando las rutas en más detalle

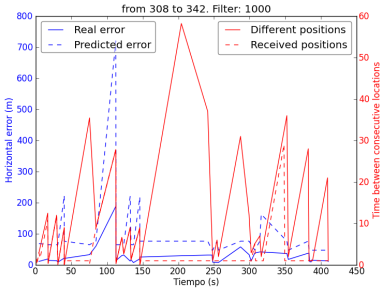
se puede observar que siete de las nueve rutas son más precisas que la original. El sistema proporciona una ruta por la que pude ser movido por el modelo un avatar sin quedar atascado para las nueve rutas analizadas.

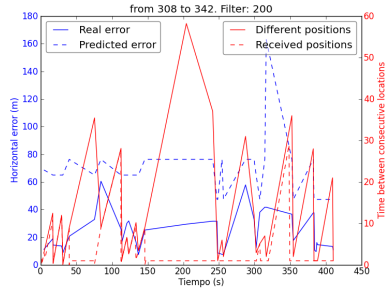
Para finalizar se presenta el sistema final. Este esta compuesto de la aplicación para iOS, el servidor web con la base de datos y el reproductor del modelo 3D. Los resultados del sistema final son muy prometedores y Utopian City_Scape ha expresado su intención de continuar con su desarrollo en el futuro.

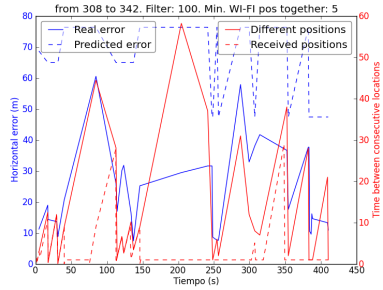
Apéndice A

Routes maps and graphs

A.1. Routes which go inside a building

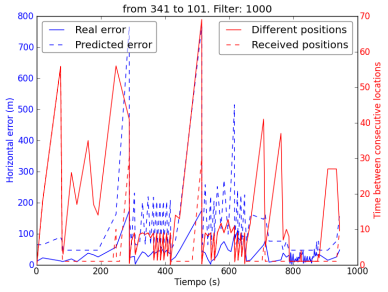


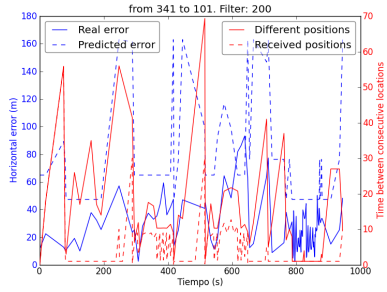


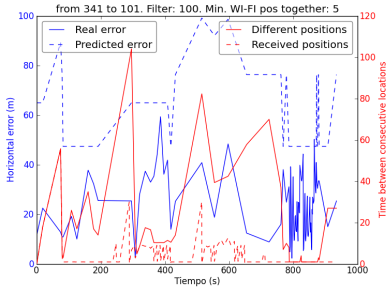


Error filter	Wi-Fi filter	Min	Max	Mean	RMSE
1000	0	7.46	188.34	26.59	38.80
200	0	7.46	60.55	23.11	26.91
150	0	7.46	60.55	22.62	26.40
100	0	7.46	60.55	22.62	26.40
100	2	7.46	60.55	22.62	26.49
100	3	7.46	60.55	22.62	26.49
100	4	7.46	60.55	22.62	26.49
100	5	7.46	60.55	22.62	26.49
100	6	7.46	60.55	24.16	28.02
100	7	7.66	60.55	24.80	29.07

Figura A.1: Route map and error and time graphs. From 308 to 342

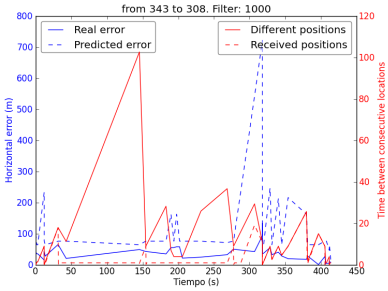


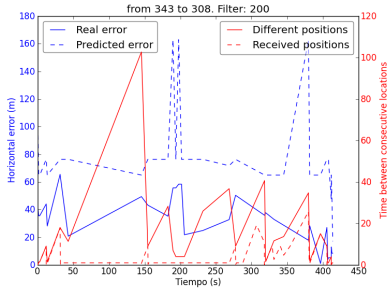


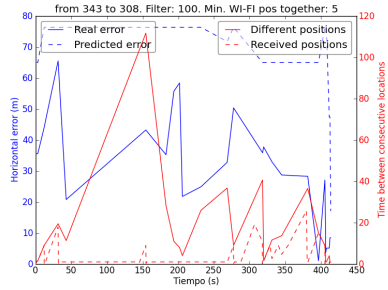


Error filter	Wi-Fi filter	Min	Max	Mean	RMSE
1000	0	2.28	175.08	30.86	40.05
200	0	2.35	93.29	26.25	31.16
150	0	2.35	93.29	24.96	29.51
100	0	2.35	93.29	24.35	28.64
100	2	2.35	93.29	24.35	28.64
100	3	2.35	93.29	24.35	28.64
100	4	2.35	59.39	22.81	25.58
100	5	2.35	59.39	22.81	25.58
100	6	2.35	59.39	22.81	25.58
100	7	2.35	59.39	22.81	25.58

Figura A.2: Route map and error and time graphs. From 341 to 101

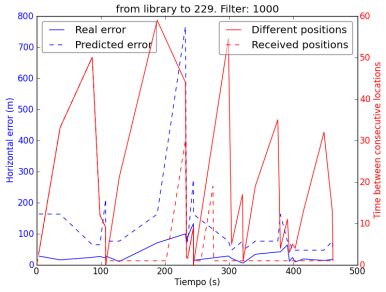


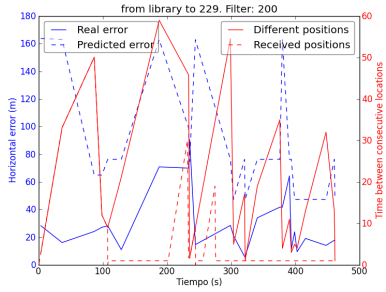


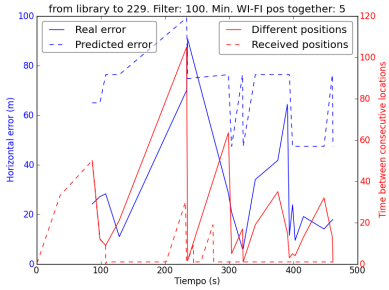


Error filter	Wi-Fi filter	Min	Max	Mean	RMSE
1000	0	1.04	133.87	36.23	44.07
200	0	1.04	65.52	31.58	36.17
150	0	1.04	65.52	30.30	34.76
100	0	1.04	65.52	29.90	34.49
100	2	1.04	65.52	29.87	34.59
100	3	1.04	65.52	29.87	34.59
100	4	1.04	65.52	29.87	34.59
100	5	1.04	65.52	29.87	34.59
100	6	1.04	65.52	29.87	34.59
100	7	3.66	65.52	30.63	35.89

Figura A.3: Route map and error and time graphs. From 343 to 308



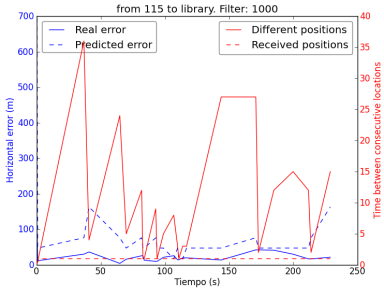


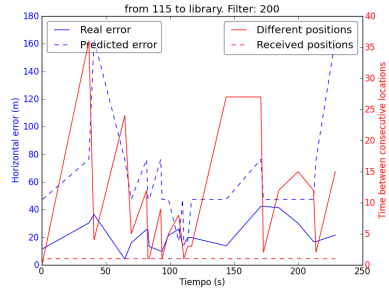


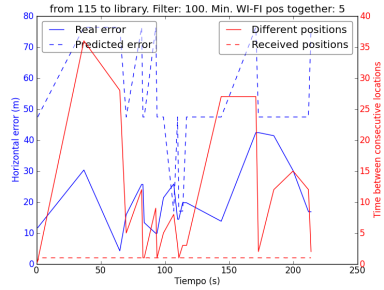
Error filter	Wi-Fi filter	Min	Max	Mean	RMSE
1000	0	6.13	132.91	35.49	47.02
200	0	6.13	90.78	29.57	36.78
150	0	6.13	90.78	28.37	35.89
100	0	6.13	90.78	28.37	35.89
100	2	6.13	90.78	28.37	35.89
100	3	6.13	90.78	28.37	35.89
100	4	6.13	90.78	28.37	35.89
100	5	6.13	90.78	28.37	35.89
100	6	6.13	90.78	28.37	35.89
100	7	6.13	90.78	28.37	35.89

Figura A.4: Route map and error and time graphs. From library to 229

A.2. Outdoor routes with curves

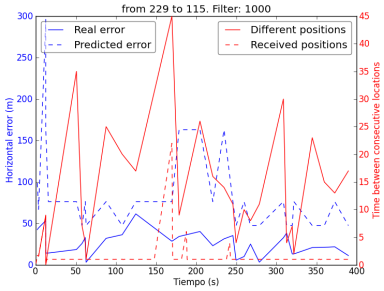


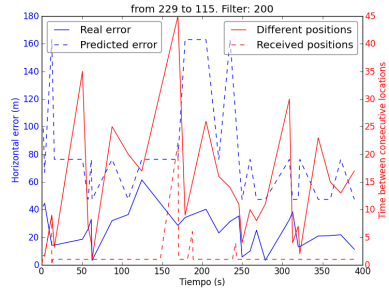


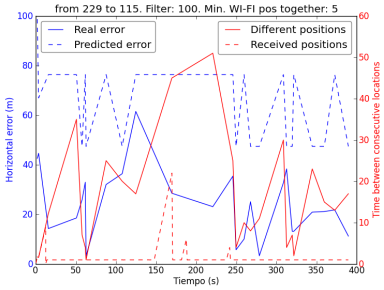


Error filter	Wi-Fi filter	Min	Max	Mean	RMSE
1000	0	4.24	99.34	24.94	30.97
200	0	4.24	42.39	21.84	24.25
150	0	4.24	42.39	21.18	23.66
100	0	4.24	42.39	21.18	23.66
100	2	4.24	42.39	21.18	23.66
100	3	4.24	42.39	21.18	23.66
100	4	4.24	42.39	21.18	23.66
100	5	4.24	42.39	21.18	23.66
100	6	4.24	42.39	21.18	23.66
100	7	4.24	42.39	21.18	23.66

Figura A.5: Route map and error and time graphs. From 115 to library

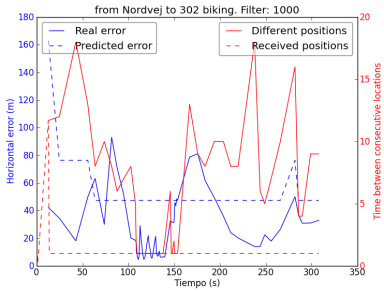


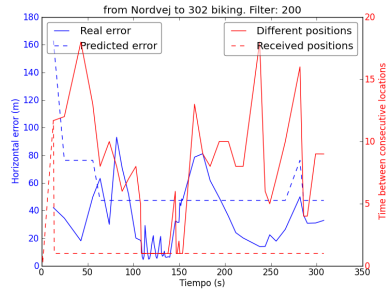


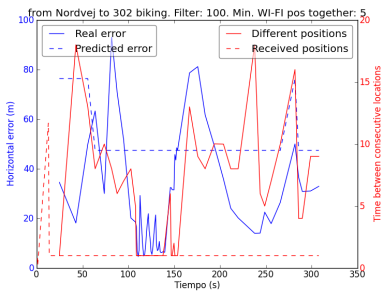


Error filter	Wi-Fi filter	Min	Max	Mean	RMSE
1000	0	3.24	61.52	26.33	29.92
200	0	3.24	61.52	25.39	28.77
150	0	3.24	61.52	24.64	28.30
100	0	3.24	61.52	24.64	28.30
100	2	3.24	61.52	24.64	28.30
100	3	3.24	61.52	24.64	28.30
100	4	3.24	61.52	24.64	28.30
100	5	3.24	61.52	24.64	28.30
100	6	3.24	61.52	24.64	28.30
100	7	3.24	61.52	24.64	28.30

Figura A.6: Route map and error and time graphs. From 229 to 115



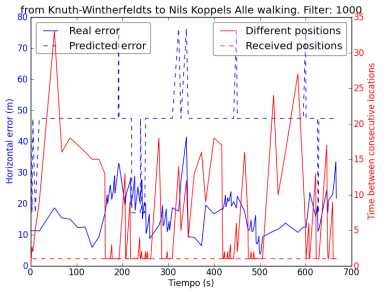


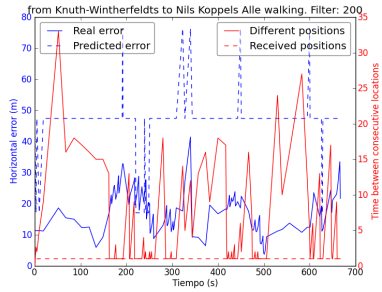


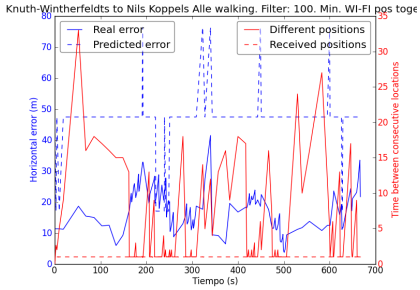
Error filter	Wi-Fi filter	Min	Max	Mean	RMSE
1000	0	4.62	93.08	26.57	33.60
200	0	4.62	93.08	26.57	33.60
150	0	4.62	93.08	26.33	33.46
100	0	4.62	93.08	26.33	33.46
100	2	4.62	93.08	26.33	33.46
100	3	4.62	93.08	26.33	33.46
100	4	4.62	93.08	26.33	33.46
100	5	4.62	93.08	26.33	33.46
100	6	4.62	93.08	26.33	33.46
100	7	4.62	93.08	26.33	33.46

Figura A.7: Route map and error and time graphs. From Nordvej to 302 biking

A.3. Straight routes

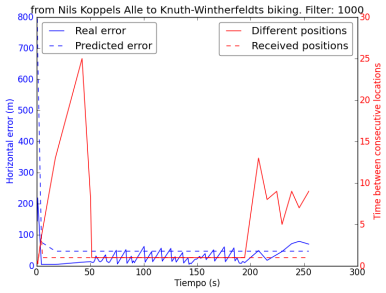


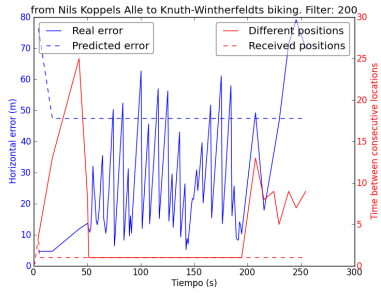


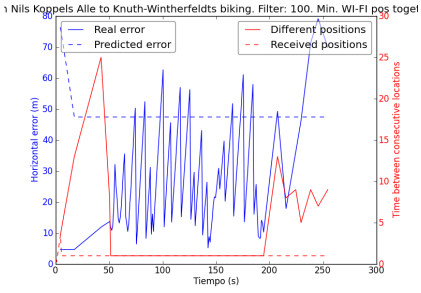


Error filter	Wi-Fi filter	Min	Max	Mean	RMSE
1000	0	3.76	41.47	17.83	19.02
200	0	3.76	41.47	17.83	19.02
150	0	3.76	41.47	17.83	19.02
100	0	3.76	41.47	17.83	19.02
100	2	3.76	41.47	17.83	19.02
100	3	3.76	41.47	17.83	19.02
100	4	3.76	41.47	17.83	19.02
100	5	3.76	41.47	17.83	19.02
100	6	3.76	41.47	17.83	19.02
100	7	3.76	41.47	17.83	19.02

Figura A.8: Route map and error and time graphs. From Knuth-Wintherfeldts to Nils Koppels Alle walking







Error filter	Wi-Fi filter	Min	Max	Mean	RMSE
1000	0	4.69	218.90	30.32	38.33
200	0	4.69	79.13	28.37	32.32
150	0	4.69	79.13	28.37	32.32
100	0	4.69	79.13	28.37	32.32
100	2	4.69	79.13	28.37	32.32
100	3	4.69	79.13	28.37	32.32
100	4	4.69	79.13	28.37	32.32
100	5	4.69	79.13	28.37	32.32
100	6	4.69	79.13	28.37	32.32
100	7	4.69	79.13	28.37	32.32

Figura A.9: Route map and error and time graphs. From Nils Koppels Alle to Knuth-Wintherfeldts biking

Apéndice B

Definition of DTU local coordinate system

BLOM
IMAGING THE WORLD

Danmarks Tekniske Universitet
Plan & Projekt
Nils Koppels Allé, Bygning 402
2800 Lyngby

10. december 2010

Att.: Markus Lampe

Definition af DTU's lokale koordinatsystem

DTU's lokale koordinatsystem (DTU-LOK) defineres i dette skrift med udgangspunkt i UTM, zone32, Euref89 koordinatsystemet (UTM). Dvs. herunder beskrives hvilke forskelle der er imellem de to systemer og hvorledes der konverteres imellem dem.

Der er tilknyttet to CAD filer til denne definition:

- **DTU_Def_in_UTM:** Sammenligning mellem DTU-LOK og UTM vist i UTM.
- **DTU_Def_in_DTU:** Sammenligning mellem DTU-LOK og UTM vist i DTU-LOK.

1.1. Koordinatsystemet

DTU-LOK er et venstrehåndssystem, dvs. X-aksen er positiv til venstre og y-aksen ligger til højre, hvilket er det modsatte af UTM og andre matematiske koordinatsystemer. Vinkler angives i DTU-LOK positivt med uret og solen. I UTM er vinklerne angivet positivt mod uret (og mod solen) – også kaldet positiv omløbsretning.

I praksis betyder det at x koordinater angivet i DTU-LOK skal ganges med -1, for at de kan anvendes i systemer, som anvender matematiske koordinatsystemer.

Se CAD filerne.

1.2. Akse retning

I DTU-LOK er y-aksen lagt parallelt med K.Wintherfeldt Allé og x-aksen langs Anker Engelundsvej. UTM har Y-aksen nord/syd.

Det betyder, at der er en drejning mellem de to koordinatsystemer. Drejningen fra UTM til DTU-LOK er på -14,3472 grader.

1.3. Nulpunkt

DTU-LOK's nulpunkt ligger i UTM koodinaterne: (N, E)=(6.187.824,989; 720.784,976)

1.4. Skalafaktor

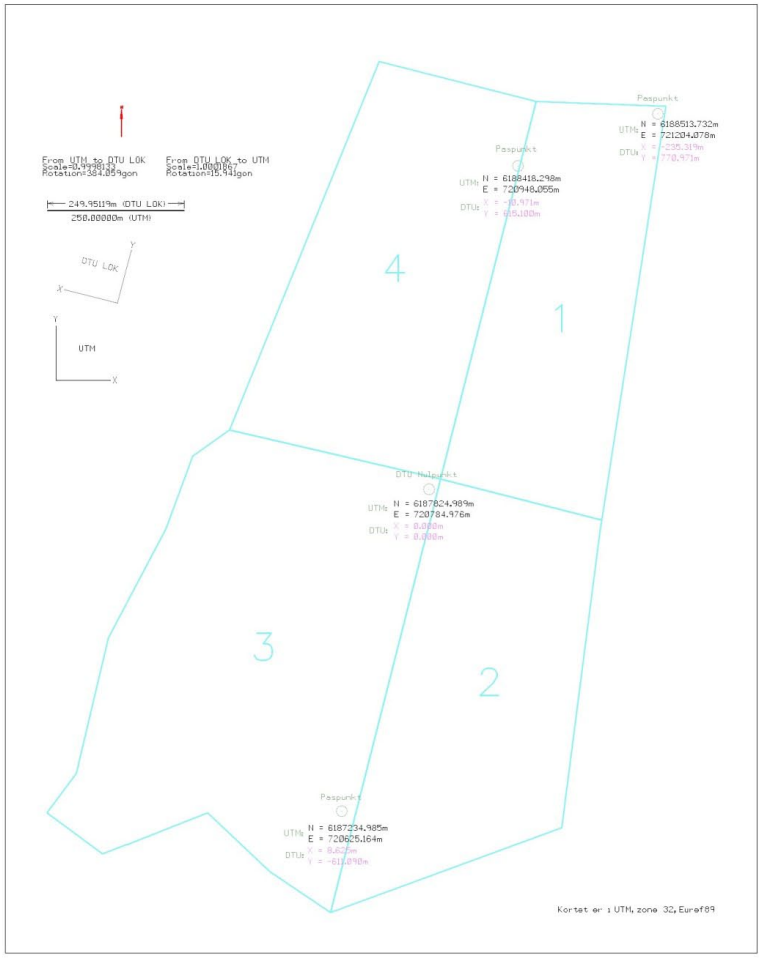
Pga. definitionen af UTM er der en skalafaktor mellem de to koordinatsystemer. Denne faktor er fra UTM til DTU-LOK på 0.9998133 og 1.0001867 når der konverteres den modsatte vej.

BlomInfo A/S
Masnedøgade 20
2100 København Ø
Tlf. 70 200 226
Fax: 70 200 227
E-mail: info.dk@blomasa.com
Web: www.blominfo.dk
CVR.: 19 95 93 41

1.5. Transformation

Til transformation mellem de to koordinatsystemer anbefales det at anvende en Plan Helmert transformation. Følgende punkter kan anvendes:

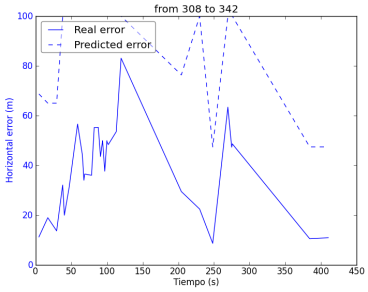
Nr.	E (UTM32, Euref89)	N (UTM32, Euref89)	X (DTU-LOK)	Y (DTU-LOK)
101-02	720625.164	6187234.985	8.625	-611.09
108-02	720948.055	6188418.298	-10.971	615.101
100	720784.976	6187824.989	0	0
1	721204.078	6188513.732	-235.319	770.97



BlomInfo A/S
Masnedøgade 20
2100 København Ø
Tlf. 70 200 226
Fax: 70 200 227
E-mail: info.dk@blomasa.com
Web: www.blominfo.dk
CVR.: 19 95 93 41

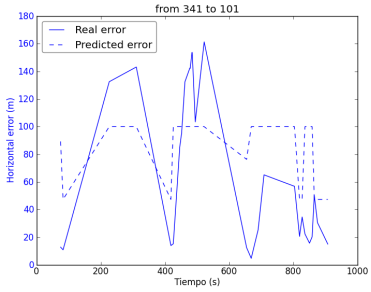
Apéndice C

Routes maps and graphs after executing the algorithm



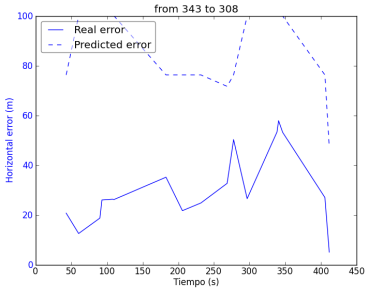
Improved with the algorithm	Min	Max	Mean	RMSE
Yes	6.54	75.48	29.97	35.07
No	7.46	32.91	17.58	19.36

Figura C.1: Route map and error graphs. From 308 to 342



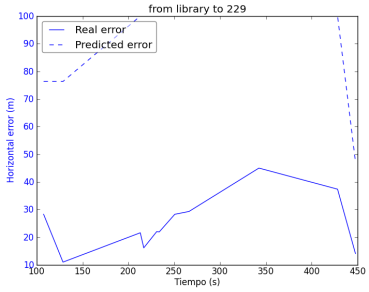
Improved with the algorithm	Min	Max	Mean	RMSE
Yes	4.76	143.17	49.70	63.36
No	2.35	59.39	22.53	25.38

Figura C.2: Route map and error graphs. From 341 to 101



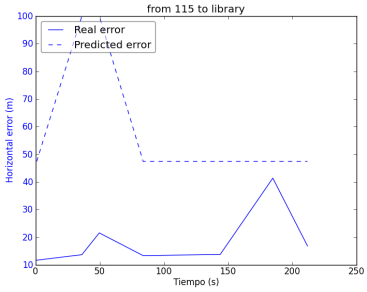
Improved with the algorithm	Min	Max	Mean	RMSE
Yes	5.14	37.15	24.39	25.75
No	1.04	50.37	26.85	30.03

Figura C.3: Route map and error graphs. From 343 to 308



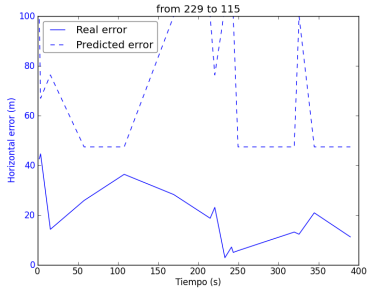
Improved with the algorithm	Min	Max	Mean	RMSE
Yes	11.01	45.02	25.02	26.81
No	9.55	90.78	31.35	39.52

Figura C.4: Route map and error graphs. From library to 229



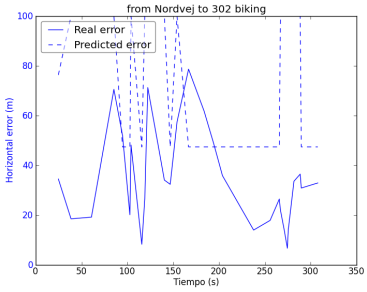
Improved with the algorithm	Min	Max	Mean	RMSE
Yes	11.70	41.39	18.91	21.23
No	4.24	41.39	19.84	22.23

Figura C.5: Route map and error graphs. From 115 to library



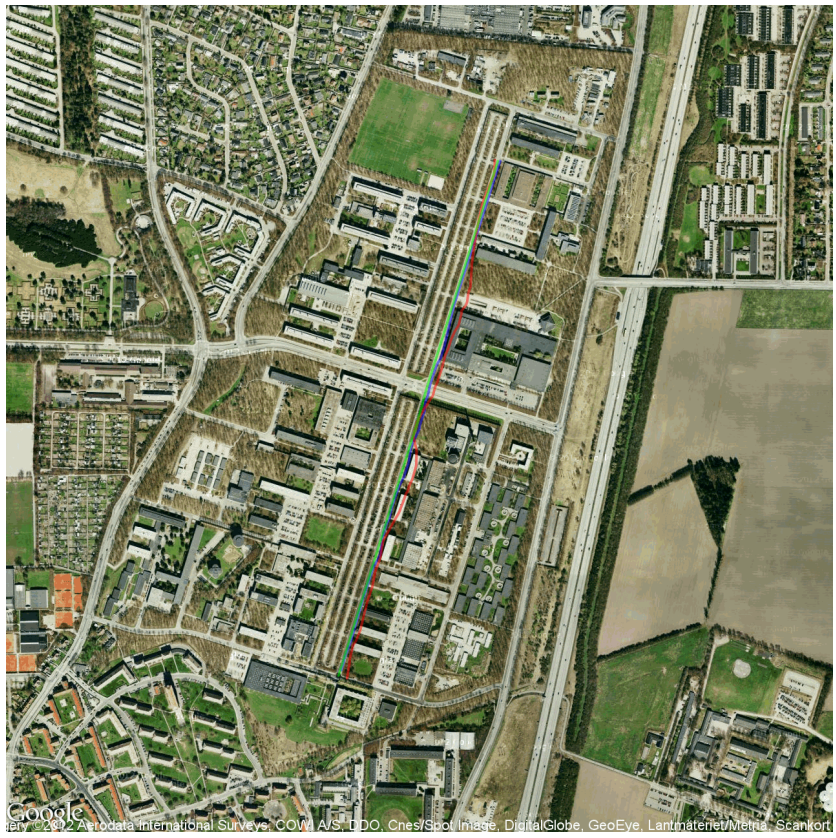
Improved with the algorithm	Min	Max	Mean	RMSE
Yes	2.89	44.64	18.70	22.66
No	3.24	44.64	21.72	25.86

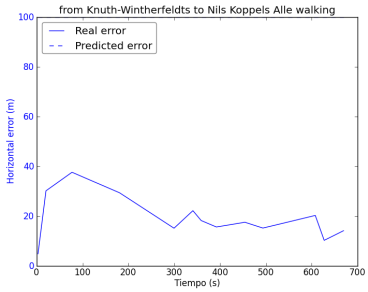
Figura C.6: Route map and error graphs. From 229 to 115



Improved with the algorithm	Min	Max	Mean	RMSE
Yes	6.70	78.68	34.58	39.67
No	4.62	93.08	26.08	33.27

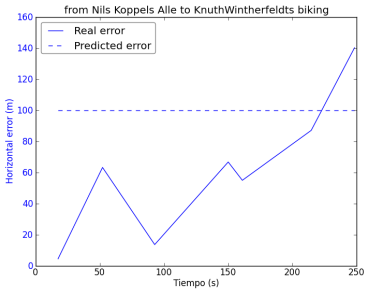
Figura C.7: Route map and error graphs. From Nordvej to 302





Improved with the algorithm	Min	Max	Mean	RMSE
Yes	4.85	37.63	19.29	21.07
No	3.76	33.49	17.74	18.86

Figura C.8: Route map and error graphs. From Knuth-Wintherfeldts to Nils Koppels Alle



Improved with the algorithm	Min	Max	Mean	RMSE
Yes	4.52	140.30	61.55	74.63
No	4.69	79.13	28.53	32.42

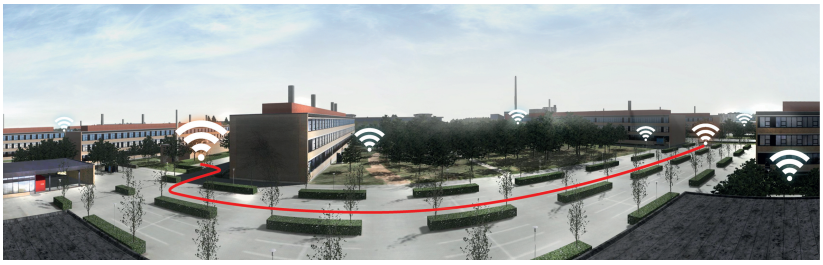
Figura C.9: Route map and error graphs. From Nils Koppels Alle to Knuth-Wintherfeldts

Apéndice D

Memoria en inglés

Tracking People Using IPhone To Locate Their Avatars In The Virtual World

Jorge Pérez Lahera



Kongens Lyngby 2012
IMM-M.Sc.-2012-80

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk IMM-M.Sc.-2012-80

Summary (English)

The goal of the thesis is to develop a tracking system using an iOS device and a 3D model mainly in the DTU campus area. The main objective of the system is that the users will switch on the app when they are located in one of the available 3D models. The app will send the users location to an Internet server. The 3D models are available through an Internet website (realsite.dk) where the user avatars will be represented tracking the user in real time.

Smartphones GPS sensors are usually not very accurate. To develop the best algorithms for the tracking with the best accuracy possible, the device accuracy has to be analyzed and measured. That is why this thesis starts with an extensive study of this sensor and the parameters that can be configured in the iOS location services. All the main problems that were solved during the development of this thesis are presented in the following sections.

The two most important parts of the proposed system have been completely developed as part of this master thesis. The first part is the iPhone app, which obtains and filters the positions of the users. Moreover, it allows the users to upload a picture to be shown in their avatars. The other developed part is an algorithm to represent the avatars in the 3D model; the row GPS measurements cannot be placed directly since they are not accurate enough. Some of the interesting improvements are that the system is able to detect when the user is in an indoor or outdoor position. It leads the avatar to the door of the building when they go inside or outside. Moreover, it detects when there is a big altitude change; the avatars use the stairs in these cases.

Summary (Danish)

Formålet med denne afhandling er at udvilke et sporingssystem, ved hjælp af en iOS-enhed og en 3D model af DTU Campus. Idéen med systemet er at brugerne tænder for app'en, når de befinder sig inden for et område med en tilgængelig 3D model. App'en sender brugernes placering til en Internet server. 3D modellerne er tilgængelige gennem en hjemmeside (realsite.dk), hvor brugernes avatarer vil blive brugt til at spore brugerne i realtid.

GPS sensorer i Smarphones er normalt ikke ret præcise. For at udvikle den bedste algoritme til sporing, med den bedst mulige præcision, er det nødvendigt først at analysere nøjagtigheden af enheden. Derfor begynder denne afhandling med en omfattende undersøgelse af sensoren og de parametre, der kan konfigureres i iOS placeringsservices. Alle de problemer der blev løst i forløbet med denne afhandling, bliver præsenteret i de efterfølgende afsnit.

De to vigtigste dele af det forslåede system er fuldt implementerede, som en del af denne afhandling. Den første del er selve iPhone app'en, som henter og filterer brugernes placeringer. Derudover giver det brugerne mulighed for at uploade et billede, der bliver brugt på deres avatar. Den anden implementerede del er en algoritme til at repræsentere brugernes avatar i 3D modellen; de rå GPS målinger kan ikke placeres direkte, da de er meget upræcise. Nogle af de interessante forbedringer er, at systemet kan opdage om brugeren er indendørs eller udendørs. Det leder avataren til en dør i bygningen, hvis de bevæger sig indendørs eller udendørs. Systemet kan også tage højde for hvis der er stor højdeforskel; i det tilfælde bruger avataren trapperne.

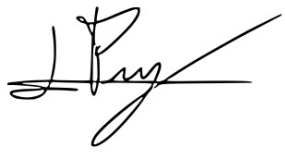
Preface

This thesis was prepared at the department of Informatics and Mathematical Modelling at the Technical University of Denmark in fulfillment of the requirements for acquiring an M.Sc. in Telecommunications engineering. The thesis was made in collaboration with Utopian city_scape as part of the Erasmus exchange program during the Spring semester of 2012. It was supervised by Michael Frederiksen, Sune Lehmann and Jakob Eg Larsen.

The thesis deals with high accuracy tracking systems using smartphone's GPS sensor.

The thesis consists of a prototype development of a tracking system using a 3D model and a smartphone. The system collects the locations of the users using the smartphone GPS sensor, improves them and represents the avatars of the users in the 3D model.

Lyngby, 31-July-2012



Jorge Pérez Lahera

Acknowledgements

I am grateful to all the people at Utopian city_scape for letting me do the master thesis in their offices. They guided and supervised me during all the process. I would like to thank my supervisors in DTU and University of Zaragoza for helping my in the most difficult moments. Without them everything would have been different. Thank to my family and friends who have supported and encouraged me during all this period.

Contents

Chapter 1

Introduction

This master thesis was carried out in the company UCS (Utopian city_scape). This company has been developing 3D models of building complex during the last five years. Examples of those are the 3D model of the DTU campus or the Carlsberg Brewery. The main objective of the company is to develop interactive 3D models using gaming technologies. The 3D models are accessible using the company website which is called realsite.dk. But now UCS wants to go an step forward and connect these models with the real world taking advantage of the extensive proliferation of the smartphones. Almost every smartphone has a GPS, which allows mobile applications (named apps for the rest of the document) to obtain the user locations; this has raised the amount of location-based services in the market. The purpose of this master thesis is to develop a system that will obtain the position of users carrying an iOS device and will represent it in a 3D model of the corresponding area. All the test performed as part of this work have been developed in the DTU campus, since UCS developed an accurate 3D model of this area.

1.1 Motivation

2011 was the first year in which vendors shipped more smartphones than PCs[1]. According to this Catalyst research, the number of smartphones and tablets shipped were 488 million while the netbooks and desktops were 414 million. These mobile devices contain a huge number of sensors such as GPS, accelerometer and compass. Moreover, they allow the user to be connected to the Internet all the time using the cellular networks. The proliferation of these mobile devices is changing the way the users connect to the Internet, new services have been created using this platform in the last years which were unthinkable few years ago. This is producing the biggest change in the Internet since the web 2.0, and the introduction of new services that were unthinkable five years ago. That is why many experts are stating that the future is just mobile. Examples of this trend can be seen in the recent Facebook acquisition of Instagram for 1 billion dollars[2]. Instagram was a two years old company with 15 employees and no revenue. But it had more than 30 million users in iOS and huge traction. This acquisition reinforces the idea of how import the mobile devices are nowadays and how important are going to be in the future.

Although there are 953 million smartphone subscribers worldwide, the smartphone user adoption has huge upside, since the number of mobile phone subscribers is 6.1 billions, as can be see in Figure 1.1. Moreover, as it is shown in the same figure, the app market is a 12 billion dollars market growing 150% a year. All these numbers motivate the author of this master thesis and UCS to develop a system using smartphones. The development tools are very powerful, which implies that the development time is sorter than using other platforms. This allows companies like UCS to develop advanced services that using other devices would have been much more expensive and maybe unattainable. One of the reasons of the big success of smartphones is the fact that they create big business opportunities for small companies such as UCS. Furthermore, these numbers are going to keep growing in the future. As can be seen in Figure 1.2, the time that the users spend in the different medias and the advertising expending are not evenly distributed. The percentage of advertising invested in the Internet and in mobile devices is expected to grow in the future to compensate the time that the users spend in these medias increasing the revenue in these areas.

As described before, the smartphone business is a huge market that is growing very fast and it is in the beginning of its expansion. The two more spread

[1] http://www.engadget.com/2012/02/03/canalys-more-smartphones-than-pcs-shipped-in-2011/

[2] http://www.forbes.com/sites/bruceupbin/2012/04/09/facebook-buys-instagram-for-1-billion-wheres-the-revenue/

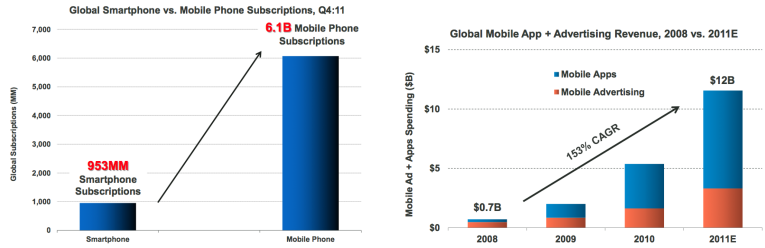


Figure 1.1: Global Smartphone vs Mobile Phone Subscriptions (left), mobile apps revenue (right)[a]

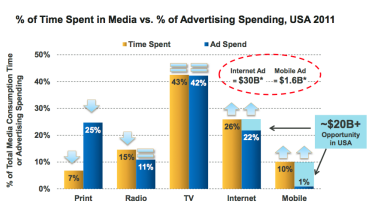
[a] Internet trends at the All Things Digital conference `http://www.forbes.com/sites/reuvencohen/2012/05/30/top-internet-trends-for-2012-according-to-vc-firm-kleiner-perkins-caufield-byers/`

smartphone operating systems are iOS and Android. As it was stated in the comScore Reports April 2012, the market share of iOS and Android in U.S. is 50.8% and 31.4% respectively[3]. They are the only mobile operating systems whose market share is growing nowadays. The number of apps available is astonishing in both platforms, more than 500.000 and more than 400.000 in iOS and Android respectively, as it is shown in Figure 1.2. The apple app store crosses 25 billion downloads in March 2012[4]. All these numbers indicates that there are not only many smartphone users worldwide but also that they use apps on a regular bases. The operating system selected for the mobile application that was developed as part of the system of this master thesis is iOS. The selection was made based on the available devices, which made easier to develop for this platform. UCS wants to expand the system into Android devices in the future.

Every modern smartphone has a GPS sensor, a Wi-Fi receiver and is connected to the cellular networks. This allows the phone to obtain the position of the user very fast and with different levels of accuracy and battery consumption using a hybrid positioning system. Since the point of view of the developer obtaining the position of the user is really straight forward using high level libraries. But the location services have still problems nowadays, if the application needs robust accuracy. It cannot provide better accuracy than 5 – 10 m in many cases, as it is demonstrated in this master thesis. This is a big problem for high accuracy tracking systems like the one developed for this master thesis. This should improve in the future with higher sensibility GPS chips and maybe using other

[3] `http://www.comscore.com/Press_Events/Press_Releases/2012/6/comScore_Reports_April_2012_U.S._Mobile_Subscriber_Market_Share`

[4] `http://www.engadget.com/2012/03/03/apple-app-store-25-billion/`



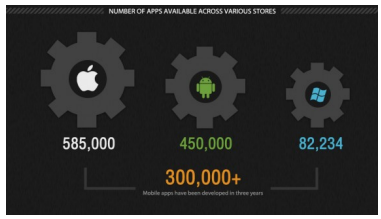


Figure 1.2: % of time spent in media vs % of advertising spending (left)[a], number apps available across various stores (right)[b]

[a] Internet trends at the All Things Digital conference `http://www.forbes.com/sites/reuvencohen/2012/05/30/top-internet-trends-for-2012-according-to-vc-firm-kleiner-perkins-caufield-byers/`

[b] `http://visual.ly/world-mobile-apps-0`

systems such as Galileo[5].

The proliferation of the smartphones is bringing the population of LBS (location based services) such as the system developed in this master thesis. Examples of this are the social network Foursquare[6] or Facebook places[7], which allows users to share their locations. Another example is the app that Apple release with iOS 5 that is called "Find My Friends"[8] which shows the locations of your friends. But these three examples cannot be considered as high accuracy tracking systems because the locations are updated on demand or at low frequency rate. The most common apps that perform high rate tracking are sport monitoring ones. The three most popular ones in this area are Endomondo[9], Runkeeper[10] and Runtastic[11]. Another app that does tracking of the user to share their positions in real time is Glympe[12]. These four examples perform similar actions than the app that was developed as part of this master thesis; they obtain the locations of the user and send it to an Internet server in real time.

However, the system developed in this master thesis, unlike all the tracking systems described before, does not use a plain map to represent the routes. It uses a high accuracy 3D model of the area where the tracking of the user was performed. This fact makes this system so innovative, no other similar system has been found. Although the great majority of the electronic maps that are

[5] `http://en.wikipedia.org/wiki/Galileo_(satellite_navigation)`
[6] `https://foursquare.com/`
[7] `http://www.facebook.com/about/location/`
[8] `http://itunes.apple.com/us/app/find-my-friends/id466122094?mt=8`
[9] `http://www.endomondo.com/login`
[10] `http://runkeeper.com/`
[11] `http://www.runtastic.com/`
[12] `http://glympse.com/`

usually used in computers or smartphones are plain such as Google Maps or OpenStreetMap; the use of 3D maps is becoming more popular in the recent years. Google maps has a 3D version of the most important cities in the world. Moreover, other companies such as Upnext produce 3D maps of big cities, which are available using their mobile application. Apple bough two companies that produce 3D maps which are Poly9 and C3 Technologies in July 2010 and October 2011 respectively[13]. Last June, Apple announces that the new version of iOS, iOS 6, will bring a new Apple made 3D maps[14]. An example of this 3D map made by Apple can be seen in Figure 1.3 All these movements just indicate that the 3D maps are the future of digital mapping.

Figure 1.3: New 3D maps in iOS 6[a]

[a] http://www.apple.com/ios/ios6/#maps

UCS has been producing interactive 3D maps for more than five years. Tracking users using these 3D models will produce a novel digital representation of the reality that have never done before. This master thesis is a proof of concept of this system. Although the system does not have a direct useful implementation yet, UCS and the author of this master thesis strongly believe in the usefulness of this system in the future. The combination of the 3D models with the tracking of users using smartphones is a field that has to grow in the future with the popularization of 3D maps. The use of high accuracy 3D maps adds additional information to the tracking of the users which can be used to improve the

[13] http://9to5mac.com/2011/10/29/apple-acquired-mind-blowing-3d-mapping-company-c3-technologies-looking-to-take-ios-maps-to-the-next-level/

[14] http://techcrunch.com/2012/06/11/goodbye-to-google-maps-with-street-view-hello-to-apples-new-maps-with-3d-flyovers/

accuracy as it is demonstrated in this master thesis. The system can differentiate where the buildings and the terrain are, therefore the user can be placed in each moment in the correct place. An accurate altitude can be obtained for each position unlike the one that the Location API obtains which usually has an accuracy of more than 20 m. Other information that can be easily obtained is where the roads are, the slope of a surface, where the doors of the buildings are. All this information combined with intelligent designed algorithms can transform really inaccurate routes into feasible routes that can be represented in a high accurate 3D model as it is done in this master thesis.

1.2 Collaboration with Utopian City_Scape

This master thesis has been developed in collaboration with Utopian City_Scape. UCS stated at the beginning the general requirements of the project. The whole master thesis has been an innovation process in which the initial project has evolved until the final result presented in this report. During all this evolution the pursuit of each decision has been to find something useful and interesting for both parts, the company and the student.

During the entire project the student has collaborated with the people of UCS. They supervised and guided the work that the student was doing. The student developed entirely the mobile application and the algorithm to improve the routes. The UCS workers joined this algorithm with the company website (realsite.dk) and did a web application which allows the app to store the routes in the server database.

1.3 Project requirements

The general project requirements are to develop a system that tracks users in the DTU area to place them in the 3D model. The system should be easy to scale into other 3D models of the company and to integrate in the company's actual products. As it is demonstrated in this master thesis the developed system fulfills all this requirements.

In more detail the required modules to be developed are:

- Analysis of the location services in iOS devices.
- IOS app to acquire the locations of the user and send it to the server.

- Server to store, process and send the locations to the 3D model player.
- 3D model player to see the routes, which includes algorithms to enhance the raw routes received according to the 3D model details.

1.4 Project architecture

The final developed system is composed of three main parts, the mobile application, the Internet server and the 3D model player. The mobile application is in charge of obtaining the user locations using the GPS of the phone and sending it to the UCS server. In order to do that, the app recognizes in which 3D model the user is placed and asks for a route name. The positions that the app receives are filtered and sent to the server in real time. The server stores these locations for each route in a database. Each model has its own coordinate system; thus the server performs the transformation from geographic coordinates to the coordinates of each of these models. The 3D model player is accessible through the company website (realsite.dk), which can be accessed using a PC browser. The algorithm to improve the routes is executed in this side. Therefore, the server sends the locations to the player. The user will be represented in this player in real time when he is using the app.

1.5 Thesis structure

So far, this introduction has unveiled the motivation to do this project, the project requirements and architecture. Moreover, it explained how the collaboration with the company and the student worked during the whole project.

Chapter 2 is an accuracy study of the location services in iOS. The analysis includes not only static positions but also recordings while moving. For the static positions, the mean horizontal error for different configured accuracy levels is compared. Moreover, the estimated accuracy and the time to first-fix are analyzed. Regarding the positions recording in motion, several routes were made in the DTU campus for this analysis. The mean error and the time between different locations were studied. Furthermore, a noise filter and an indoor detector were designed which are used in the following chapters.

Chapter 3 shows the algorithms that were designed to represent the routes in the model since an avatar can not move in the model following the original route. It presents how the coordinates are converted from geographic coordinates to

DTU coordinates and vice versa. The result routes are analyzed and compared with the original ones. To conclude this section, possible future improvements of the algorithm are discussed.

Chapter 4 presents the final system integrated with realsite.dk.

Chapter 5 summarizes the whole report and emphasizes the most important results presented.

Chapter 2

Accuracy study of iOS location services

2.1 Introduction to the different location systems

In this first section, the different ways that smartphones are used to get the location of the user is going to be briefly explained. All the modern smartphones use a hybrid location system, this means that three different positioning technologies can be used to obtain the user locations. These systems are satellite positioning (A-GPS), Wi-Fi positioning and cellular positioning.

2.1.1 Satellite positioning

Satellite positioning is a technology in which several satellites are used to estimate the user location. The most well known satellite-positioning system nowadays is GPS (Global Positioning System). The GPS system was designed by the United States Department of Defense (DoD), has been in Fully Operational Capability (FOC) from July 1995. It was originally composed of 24 satellites but currently it consists of 32[1]. This assures that a mobile device equipped

[1]GPS Constellation Status `ftp://tycho.usno.navy.mil/pub/gps/gpstd.txt`

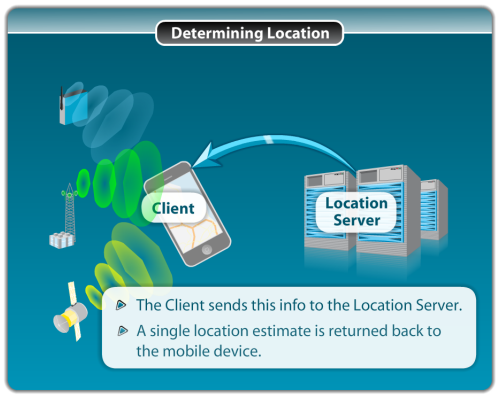


Figure 2.1: Example of hybrid positioning system[a]

[a] `http://www.skyhookwireless.com/howitworks/loader_howitworks.swf`

with a GPS receiver will catch up signals from at least four different satellites [ZA06][ML08].

The GPS satellites broadcast two signals in the L1 (1575.42 MHz) and L2 (1227.60 MHz) bands, which broadcast time and orbital information; the locations are being calculated through this information. Two services are provided: the Standard Positioning Service (SPS) and the Precise Positioning Service (PPS). SPS is the one that is used by the entire end costumer GPS enabled products and is the one that is used in the devices analyzed in the project, while PPS offers enhanced accuracy but it can only be used by authorized users for instance the US army [Hub].

The majority of GPS-enabled smartphones, including all models of iPhone, employs a technology, which is called Assisted GPS (A-GPS) [GS05]. With this technology a remote GPS location server performs many functions that usually are performed in a full GPS receiver. This server provides the A-GPS device with satellite orbit and clock information and position computation. The mobile device does not need to decode the GPS messages or perform a search for visible satellites when the system is starting, the server assists the device in these tasks. These improve the power consumption and the time-to-first-fix [ML08] over the traditional GPS receivers [Zan09].

The signal propagation errors limit the GPS accuracy. These are the typical errors that may be encountered in all satellite communications. The signal in the path from the satellite to the mobile device does not have a constant speed,

this produces a inconsistent propagation delay. Moreover, the signal bounces off obstacles before arriving to the device antenna; this produces signal quality degradation due to multipath fading errors.

Neither GPS nor A-GPS works well in high-density urban areas and in indoor locations due to poor satellite visibility and signal attenuations. High-sensitivity GPS (HSGPS) chip sets try to solve this problem and are implemented in many new A-GPS receivers. But even with this chipsets there are many locations where the device is not able to find the current position.

Hence, other positioning systems apart from GPS have to be implemented in smartphones if the user has to receive a position in all the locations. Almost all smartphones deploy Wi-Fi positioning and cellular positioning in addition to A-GPS, which are explained in the following sections.

2.1.2 Wi-Fi positioning

This technology uses WiFi access points (APs) to determine the user location. Over the last several years millions of wireless networks have been deployed everywhere. Nowadays, it is hard to find a place inside a city where there is no WiFi APs available, a study about APs density can be found in [JL07]. Each AP has a unique identifier that is called BSSID which is the MAC address of the AP and in order to be visible for other devices beacon frames are sent periodically. These facts make WiFi access a very appropriate location system in the places where A-GPS is less accurate or even can not give a location which is in high density urban areas and indoor locations. Thus, the WiFi positioning enabled device records BSSID and signal strength of all the available networks at a particular location to obtain the locations. This allows encrypted and weak signals to be used.

As it is described in [Kue05], the WiFi positioning algorithms can be upstream or downstream and are divided into three main categories: proximity sensing, lateration and fingerprinting. In proximity sensing the position of the AP with best signal strength is adopted. Lateration method tries to measure the distance between the AP and the device using the path loss experienced by the beacon packets during transmission. While with fingerprinting, the set of APs and the signal strengths for these APs that the device receives in a location ("fingerprint") is compared with patterns that have been measured before at known locations. The pattern that is more similar to the fingerprint that the device is measuring meanwhile is adopted as the actual position [Hub].

Fingerprinting technologies have two big advantages over the other techniques.

They do not require the exact location of the APs and do not try to model signal strength. These advantages have made fingerprinting the preferred technology for big scale WiFi positioning in metropolitan areas. In order fingerprinting technologies to be available in an area, the WiFi signals have to be observed at known locations in that area, this is called calibration phase or offline phase, as it can be seen in Figure 2.2. During this phase, a WiFi receiver is hooked up to a GPS device and the WiFi signals are recorded with the GPS positions as the device moves through an area. This WiFi fingerprints for known locations are stored in a database and compared with the signals that the device receives when it is in an unknown location, the location is determined finding the closest match, this phase is called positioning or online phase. There are several matching techniques that have been developed for this, but the most widely used is the K-nearest neighbor estimation because of its computational simplicity and performs well in contrast with other techniques [Zan09] [SC05].

Figure 2.2: Collection of data for WiFi and Cellular positioning[a]

[a] http://www.skyhookwireless.com/howitworks/loader_howitworks.swf

Several WiFi positioning systems are currently in the market, the most remarkable one is the one that is created by Skyhook Wireless which is used by ample well known companies. It was also used in iOS until April 2010. Other alternatives are Navizon and PlaceEngine. Moreover, Apple and Google have their own WiFi positioning system[2] [3].

[2] In April, Apple Ditched Google And Skyhook In Favour Of Its Own Location Databases http://techcrunch.com/2010/07/29/apple-location/

[3] Copy of Google's submission today to several national data protection authorities on vehicle-based collection of wifi data for use in Google location based services http://static.googleusercontent.com/external_content/untrusted_dlcp/www.google.com/en//googleblogs/pdfs/google_submission_dpas_wifi_collection.pdf

2.1.3 Cellular positioning

It is more than 20 years since the first GSM call, made by Finnish Prime Minister Harri Holkeri[4]. Since then, a huge expansion of cellular networks has been developed in the world, to an extent of having almost worldwide coverage nowadays. The two main cellular networks that are in operation in the world are The Global System for Mobile Communication (GSM) and the Universal Telecommunication System (UMTS), which are commonly called 2G and 3G networks, respectively. Moreover, the new 4G networks which is called LTE is starting its deployment but it is far from the actual coverage that 2G and 3G have, and it will take several years to achieve it. The cellular positioning techniques that are explained in this section are only applicable for 2G and 3G networks.

Cellular networks are divided into cells, which have a hexagonal shape since a theoretical point of view. Each of these cells has a base station (BS) in the middle. The mobile phone of the user is connected to one of these base stations which is usually the one that has a better signal strength and switches between them automatically. The simplest way of cellular positioning is to use the location of this BS as the user location. This method is called cell identification (cell ID). The precision of this method is conditioned by the cell size, which varies from urban to rural areas. These base stations can have mainly two different kinds of antennas, omnidirectional and directional antennas. If the antenna is directional, the cell is divided into sectors, which means that the user location can be obtained with better precision.

Other techniques have been developed for improving cell identification. One of them it is called enhanced cell ID (E-CID), this method measures the time that the signal takes to arrive from the device to the base station which is measured by the base station and is called time advance. The mobile device is usually within the range of multiple base stations; several techniques try to take advantage of that. These techniques are Time Difference of Arrival (TDOA) or Angle of Arrival (AOA) [Kue05]. Same as for WiFi positioning, fingerprinting techniques have also been developed [SB04] and [CMYA06].

Although the accuracy of cellular positioning is lower than GPS or WiFi, the availability of the cellular networks are much higher. Moreover, it is very energy efficient because it uses data and hardware that it is available all the time in the phone, the phone is always connected to the cellular network. Thus, it does not need to switch on any other hardware like GPS or WiFi.

[4]GSM turns 20 today, still rocking the world `http://www.engadget.com/2011/07/01/gsm-turns-20-today-still-rocking-the-world/`

2.2 Location services in iOS

The iPhone 3G, which was released on 11 July 2008[5], was the first commercial device presenting the hybrid positioning system [Zan09]. But, nowadays almost all the smartphones implement this technology. As already mentioned, this system acquires the user location choosing the best option for each situation between satellite, WiFi or cellular positioning. This is done automatically by the operating system; neither the users nor the developers have to care about it. In previous versions of iOS the Maps app shows different pins depending on the type of location that it was used but in the recent versions (currently 5.01) all the positioning systems show the same pin and the only clue that can be used to differentiate them is the accuracy that the location services provides.

Until April 2010, Apple relies on Skyhook and Google data to provide WiFi and Cellular positioning systems respectively. But starting with iOS 3.2, which was released in April 2010, Apple uses its own databases to provide location-based services (LBS)[6]. As Apple states in the official documentation, this database is maintained by iOS users.*"If Location Services is on, your device will periodically send the geo-tagged locations of nearby Wi-Fi hotspots and cell towers in an anonymous and encrypted form to Apple, to augment the crowd-sourced database of Wi-Fi hotspot and cell tower locations."*[7].

After this short introduction about the location services in iOS from now until the end of this section, the technical details and the parameters that the location API allows the developer to configure will be explained. The only way the developer can access the positioning features in iOS is using the Core Location API provided by Apple. This API provides the location using the hybrid positioning system but the developer can not choose which of the three systems is going to be used directly, only the desired accuracy can be chosen.

The Core Location framework provides three different services to monitor the device's location: the significant-change, standard location and region monitoring location services. In this project, only the standard location service is going to be studied because it is the most easily configurable and thereby it can be the most accurate. For the purposes of this study accuracy is one of the main constraints because the model is very accurate, thus for this project the significant-change and region monitoring are not feasible solutions[8].

[5] iPhone 3G announced iPhone 3G announced iPhone 3G announced `http://www.tuaw.com/2008/06/09/iphone-3g-announced/`

[6] In April, Apple Ditched Google And Skyhook In Favour Of Its Own Location Databases `http://techcrunch.com/2010/07/29/apple-location/`

[7] iOS 5: Understanding Location Services `http://support.apple.com/kb/HT4995`

[8] Location Awareness Programming Guide `https://developer.apple.com/library/ios/doc`

The standard location service is available on all devices and in all versions of iOS, thus it is the most common way to get the user's location. Before using it, you have to configure the desired accuracy and the minimum distance between two locations that produce a new location in your app.

In order to start using the standard location service you have to create an instance of the class *CLLocationManager*. After that, you have to configure its *desiredAccuracy* and *distanceFilter* properties. Once this is done, assign a delegate to the object, which implements *CLLocationManagerDelegate Protocol*[9] and calls the *startUpdatingLocation* method to begin receiving location notifications. One of the experiments with the iPhone location service that are included in this master thesis tries to measure the impact that the parameter *desiredAccuracy* has in the locations that the API provides. In the Apple documentation, the next paragraph can be found:

> "When tracking changes to the user's location, the distanceFilter property can be used to filter out update messages from the location manager to it's delegate. However, such messages may still be delivered if more accurate measurements are acquired. Also, the distanceFilter does not impact the hardware's activity - i.e., there is no savings of power by setting a larger distanceFilter because the hardware continues to acquire measurements. This simply affects whether those measurements are passed on to the location manager's delegate. Power can only be saved by turning off the location manager."[10]

As can be seen, the parameter *distanceFilter* just defines a filter to filter the locations that are received before the delegate is called. That is why, for all the experiments developed in this section this parameter was set to zero. Using this configuration, the app was able to store all the locations that the location manager receives. The number of unique and equal positions and the time difference between them are going to be analyzed in the following sections because it may contain valid information, such as the user is not moving unless the vast majority of the locations that delegate receives change.

The desiredAccuracy parameter does not guarantee any accuracy as it was stated by apple in this readme:

---

umentation/UserExperience/Conceptual/LocationAwarenessPG/LocationAwarenessPG.pdf

[9] CLLocationManagerDelegate Protocol Reference https://developer.apple.com/library/ios/#documentation/CoreLocation/Reference/CLLocationManagerDelegate_Protocol/CLLocationManagerDelegate/CLLocationManagerDelegate.html

[10] LocateMe ReadMe.txt https://developer.apple.com/library/ios/#samplecode/LocateMe/Listings/ReadMe_txt.html

> "Core Location does not guarantee that a measurement matching the desiredAccuracy will be delivered. Rather, a best effort is made, and may be constrained both by the capabilities of the device and the location and environment from which it is used. ... a GPS equiped device will often provide better than 10 meter accuracy, but not underground."[10]

This fact is going to be analyzed in the Section 2.4.2 using different values for the *desiredAccuracy* property. His property belongs to the data type *CLLocationAccuracy* which is a double that "represents the accuracy of a coordinate value in meters"[11]. There are several constants defined for this property the fourth most accurate ones are *kCLLocationAccuracyBestForNavigation, kCLLocationAccuracyBest, kCLLocationAccuracyNearestTenMeters and kCLLocationAccuracyHundredMeters* which define a value of -2, -1, 10 and 100 respectively. Moreover, this property can be set to any positive double value.

Once the location services have been started, whenever a new location is available the locationManager reports it to the *locationManager:didUpdateToLocation:fromLocation:* method of its delegate[12]. The locations are stored in objects, which belongs to the class *CLLocation*. These objects have five location attributes, which are *coordinate, altitude, horizontalAccuracy, verticalAccuracy and timestamp*. The names of it are self-explanatory, thus an explanation of its meaning is not necessary. There are other two properties, which are speed and course. All these seven parameters were stored and analyzed; the results are shown in the following sections.

This report analyzes the parameters coordinates, horizontal accuracy and timestamp. But it does not analyze other ones such as verticalAccuracy, speed and course. These parameters were not used for several reasons.

The altitude and the verticalAccuracy do not provide any useful information because the 3D model has a better estimation of the altitude that the one than the GPS can provide. Moreover the vertical Accuracy was in all the cases bigger than 20 meters. As for the speed and course, their value was not available in the majority of the time. It had a value of 0 and -1 respectively. It can be assumed that this values were more time available using the new models of iPhone which are iPhone 4 and iPhone 4S. But since the system has to work in all model of iOS devices is useless develop an algorithm which needed data is not available all the time.

[11] Core Location Data Types Reference `https://developer.apple.com/library/ios/#documentation/CoreLocation/Reference/CoreLocationDataTypesRef/Reference/reference.html`

[12] CLLocationManagerDelegate Protocol Reference `https://developer.apple.com/library/ios/#documentation/CoreLocation/Reference/CLLocationManagerDelegate_Protocol/CLLocationManagerDelegate/CLLocationManagerDelegate.html`

2.3 Experimental settings

2.3.1 Analyzed devices

The analyzed devices in this study were iPhone 3GS and Garmin forerunner 405. The iPhone was chosen because it is the device where the final app will be implemented. Garmin forerunner 405 was analyzed to compare with the results that the iPhone provides especially in the movement test. Moreover, the Garmin forerunner is supposed to have better accuracy than the iPhone, as can be seen in the next sections.

2.3.1.1 iPhone 3GS

Apple released iPhone 3GS on June 19 2009, after the big success of the iPhone 3G the new model had a big expectations. The expectations were rapidly achieved, since Apple announced that 1 million units were sold during the first weekend after the release[13]. Three years later of that date, we can say that iPhone 3GS was a commercial success for apple. This device is the device that is going to be used for the development of this master thesis. The iPhone was running iOS 5 during the analysis, concretely the versions 5.01 and 5.1.

This information is not public but according to online (non official) resources, the GPS chip that the iPhone 3GS include is supposed to be Infineon Hammerhead II, the same one that the iPhone 3G included[14] [15]. A product overview of this chip can be found in the Appendix A. It claims to have a time to first fix of 1 second at 5 m accuracy and 2 m steady state accuracy. As can be seen in the analysis these values in the iPhone are way worse than Infineon claims in the product brief. However, there is not information about how Apple configures the chip or what accuracy apple provides with this chip. Thus, the provided values cannot be used as the expected accuracy for the iPhone.

[13] Apple Sells Over One Million iPhone 3GS Models http://www.apple.com/pr/library/2009/06/22Apple-Sells-Over-One-Million-iPhone-3GS-Models.html

[14] iPad 3G Vs iPhone 3GS: 3G Speed And GPS Improvements http://www.iphonehacks.com/2010/05/ipad-3g-vs-iphone-3gs-3g-speed-gps-improvements.html

[15] iPhone 3GS Teardown http://www.ifixit.com/Teardown/iPhone-3GS-Teardown/817/2

Figure 2.3: iPhone 3GS[a]

[a] http://www.apsipirkim.lt/image/cache/data/iphone/MC555_AVM-500x500.jpg

2.3.1.2 Garmin forerunner 405 CX

The other GPS enabled device analyzed was a Garmin forerunner 405, which is a GPS enabled watch, as can be seen in Figure 2.4. This watch records the GPS positions when the training is enabled. Once the training is finished, the positions that the watch recorded can be transferred to the computer and can be exported in the format files GPX and TCX. Technical information about the GPS chip that is integrated in the watch was not available.

2.3.2 Data collection and evaluation criteria

The data was collected in a different way for the different devices. While the Garmin watch allows importing the GPS positions directly, iPhone does not allow this.

Previous research in this area placed the GPS enabled device in an ideal position using a tripod and direct the device in the direction of the maximum radiation of the GPS antenna to the sky. This is not a very realistic position if we think in terms of the daily use of a smartphone. Examples of this can be seen in [Zan09]

Figure 2.4: Garmin forerunner 405[a]

[a] https://static.garmincdn.com/en/products/010-00658-10/g/cf-lg.jpg

for the iPhone or in [ZB11] for two GPS enabled smartphones. No previous research has been found in which the devices were placed in a jeans pocket that is the normal place where the users usually place the mobile phone. In all tests in this study the iPhone was placed in a front jeans pocket while the watch was in the left hand. The user was wearing a winter jacket. The purpose of this is to get the most real results that can be used in the development of real Location Based apps. The entire tests were performed inside the DTU campus area. The four different quadrants were used for the different static and movements tests. The positions used for the static tests can be seen in Figure 2.5.

Six different static locations were chosen to perform this test. Three of them were used for the comparison between the different location services accuracy in iOS and the other three for the comparison between the phone and the watch. In both devices, the locations were harvested for periods of time of two minutes. 1887 location coordinates were gathered for the accuracy comparison study and 372 for the comparison between the two devices. As for the measurements in motion, 1791 positions were harvested using iPhone and 376 using the Garmin watch.

Regarding iPhone, the data was collected using one app that was developed entirely for the purpose of this master thesis. The idea is to use part of this app for the implementation of the final one. As can be seen in Figure 2.6, this app allows the user to select the desired accuracy and start the location services. All the locations that Core Location provides to the app since the button "start GPS" is pressed until the button "stop GPS" is pressed are stored in a database in the iPhone memory using Core Data[16] [17]. This database has two tables

[16] CS 193P iPhone Application. Lecture 13: Core Data http://www.stanford.edu/class/cs193p/cgi-bin/drupal/system/files/lectures/Lecture%2013_2.pdf

[17] Core Data Programming Guide https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/CoreData/CoreData.pdf

one table that stores the routes and other table that stores each location. The routes tables include a field that indicates which positions belong to this route, also applicable for the locations table. All the information that core location provides for each position, which was commented in the section location services in iOS, is stored in the database. All the routes keep stored in the database until the app is uninstalled. As can be seen in Figures 2.6 and 2.7, another feature is that the app allows sending the locations in real time to the Utopian Cityscape server.

The second tab of the app, which is called routes, provides to the user with one list of all the routes that are stored in the database as can be seen in Figure 2.7. One new view is presented when the user tap in one of the route rows in the table, this window can be seen in the Figure 2.7. This five buttons perform different actions that were needed for the development of the master thesis. Two of them print information in the console of Xcode to be able to export the data into a text file. The two bottom ones send information to the Utopian Cityscape server. Figure 2.7 shows how the routes are plotted in a map in the app.

As for the Garmin, the routes are transferred to a computer. After that, they can be exported as GPX or TCX files. The GPX files were used since it was easier to parse it in python and both contains the same information. Using this method, only the latitude, longitude and time were obtained for each position. As it was commented before with iPhone latitude, longitude, altitude, speed, course, horizontal accuracy and vertical accuracy were obtained for each location that the API provides. It makes sense that Core Location provides much more information that the Garmin watch since Core Location has to provide functionalities for all the location based apps in iOS. The Garmin watch just has this functionality as an additional feature since you can analyze your routes in Garmin website directly.

The GPX files were parsed using gpxpy (GPX file parser)[18]. The locations were stored in a text file with known structure, thus it was easy to process it using Java and python programs.

The data that was harvested using iPhone was printed in the phone console using the buttons that were shown before and copied in a text file. Although, it is not the best way to do that, due to the limitations that the file system has in iOS it was the most effective way.

The following sections analyze the accuracy of the location services in iOS. Both static and in motion measurements are analyzed. The horizontal error is analyzed and compared with the one that is provided by the services. The change

[18] `https://github.com/tkrajina/gpxpy`

of the behavior of the locations services according to different parameters such as desired accuracy is studied as well. Moreover, the two devices are compared and the time interval, which the new locations are received, is examined. To finalize, a filter to remove the inaccurate positions is designed and evaluated.

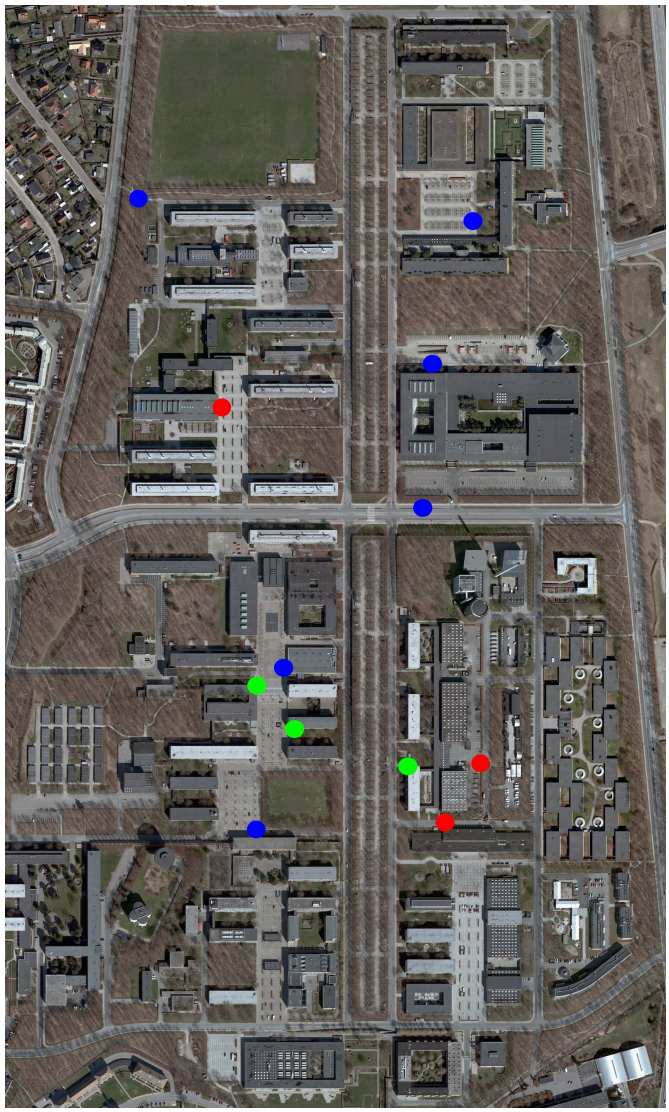


Figure 2.5: Positions where the tests were performed

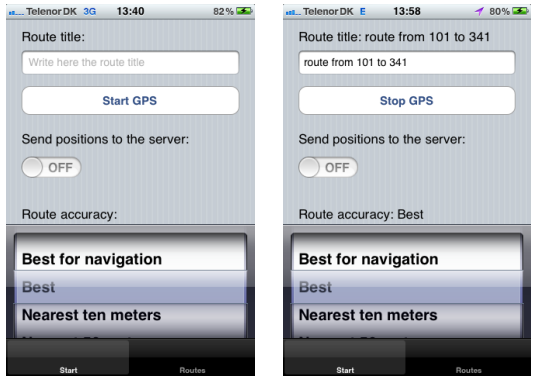


Figure 2.6: Left: initial tab screen. Right: initial tab screen while harvesting data

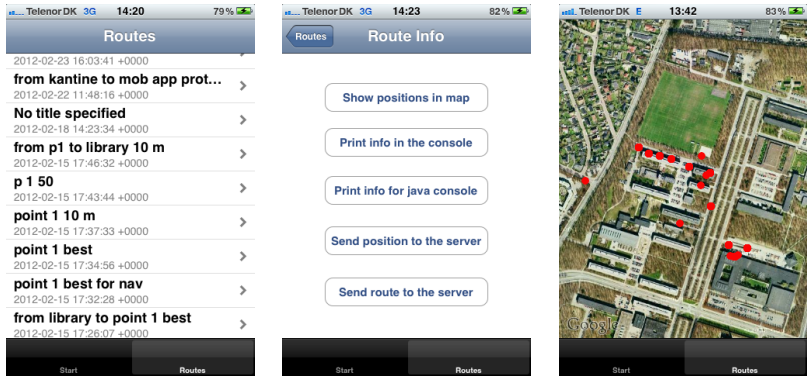


Figure 2.7: Left: routes tab. Middle: route info window. Right: map window

2.4 Analysis of Static Measurements

This section analyzes the accuracy several parameters that the location services provides in iOS. Moreover, it measures the time to first-fix and compares the accuracy of the iPhone and the Garmin watch.

2.4.1 GPS data processing

Once the data which the iPhone and the Garmin watch provides were stored in text files with a known structure, open the text file using java and python to process the locations is fairly straightforward. The first step in the calculations was to convert the geographic to UTM (Universal Transverse Mercator) coordinates.

As can be seen in the Figure 2.8, the geographic coordinates represent the locations in the hearth using three parameters, latitude, longitude and altitude. Latitude represents the angle between the equatorial plane and line perpendicular to the ellipsoid at that point. While the longitude is the angle between the meridian passing through that point and the prime meridian[19].

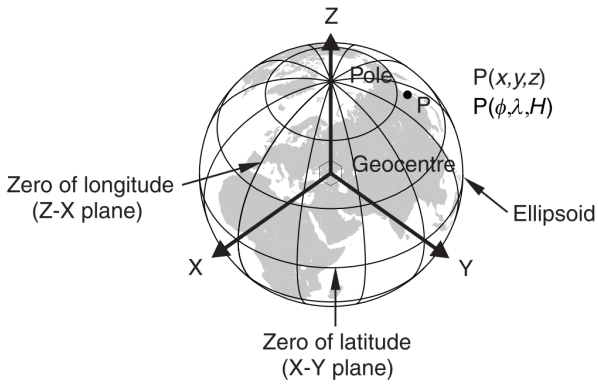


Figure 2.8: Geographic coordinates system[a]

[a]A guide to coordinate systems in Great Britain `http://www.ordnancesurvey.co.uk/oswebsite/gps/docs/A_Guide_to_Coordinate_Systems_in_Great_Britain.pdf`

[19]Geodetic Datum Overview `http://www.colorado.edu/geography/gcraft/notes/datum/datum_f.html`

At the beginning of the study, the Haversine formula was used to calculate the distance between the real coordinates that the device was and the coordinates that the location services provided, the real horizontal error[20]. But this formula calculates the distance, it does not provide any information about the direction of this error. Moreover, in order to be able to plot scatter plots with the real position and the position that GPS provides using a Cartesian coordinate system, the geographic coordinates were converted to UTM coordinates. The distance between two points in UTM coordinates is the shortest Euclidean distance in meters, this was used to calculate the horizontal error. Unlike geographic coordinates UTM is used to represent the locations in a map.

The datum used for GPS positioning is called WGS84 (World Geodetic System 1984). Thus, this is the datum that was used to convert from geographic coordinates to UTM. Further details about this conversion can be found in Section 3.3.2.1.

The java code developed for this part for each location convert all the geographic coordinates into UTM coordinates and calculate the difference in both axes (x and y) between them and the real position that the device was placed during the experiment. The real UTM coordinates were obtained using digital DTU maps that Utopian City_Scape provided which were said to be very accurate.

This program generates a new text file with the same information than the previous one but with the x and y coordinates centered in the real position instead of in the UTM coordinates system. This data was the data used to generate all graphs that area available in the results section. The graphs were generated using the python library Matplotlib[21].

2.4.2 Horizontal accuracy comparison between different desired accuracies in iOS

As aforementioned CLLocationManager allows the developer to configure the desired accuracy for the locations that provides. This parameter is a double, which represents the accuracy that the app needs in the positions that receives. For this comparison four different values for this parameter have been chosen as can be seen in the Table 2.1.

All the locations chosen for this experiment were placed in outdoor locations in the DTU area. Only 3 out of 1893 locations were obtained through Wi-Fi po-

[20] Calculate distance, bearing and more between two latitude/longitude points `http://www.ig.utexas.edu/outreach/googleearth/latlong.html#ellipsoid`

[21] `http://matplotlib.sourceforge.net/`

Table 2.1: desiredAccuracy values used in the analysis

Name of the accuracy constant	value
kCLLocationAccuracyBestForNavigation	-2
kCLLocationAccuracyBest	-1
kCLLocationAccuracyNearestTenMeter	10
(custom value)	50

sitioning and none of them were obtained using Cellular positioning. Although the Location Manager does not provide this information directly, the type of positioning system used can be guessed using the value of the horizontal accuracy. Zandbergen in [Zan09] states that if the location is obtained using Wi-Fi or cellular positioning the altitude is not reported. During this analysis, this statement has been found incorrect since for all the locations that were gathered for this analysis an altitude was provided. The previous mentioned article is from 2009, so we can think that Apple have fixed this in this three years. Doing some indoor and outdoor testing's, the horizontal accuracy that the Location Manager provides in Wi-Fi positioning has been found to be 65 while in cellular positioning it is an integer number usually bigger than 1.000. If A-GPS is used horizontal accuracy is usually smaller than 100 and it is a float number with 14 decimal positions. This can likely be attributed to the algorithm to calculate the accuracy in GPS is much more accurate that the one that it is used in the other two methods.

Therefore, this section analyzes the accuracy of the A-GPS in the iPhone and not the other location systems. Figure 2.9 shows a scatter plot of the horizontal error calculated as was explained in the previous sections. The X and Y axes correspond with the X and Y coordinates of UTM. At a first glance, it can be seen that the best for navigation, best and nearest 10 m accuracies parameters provide similar results while the nearest 50 m provides slightly worse results. Other remarkably fact that can be seen in the graph is that the great majority of the positions are placed in the surroundings of the diagonal that cross from negative X and Y to positive X and Y. Moreover there are more positions in the right part of the graph than in the left one. Since, with this data we can conclude that the GPS positions provided by iOS location services were more likely to be in the south, north and east area from the correct position than to the west. But due to the limitations of this test we can not conclude that this is going to happen in all the situations.

Table 2.2 summarizes the horizontal error calculations done for this experiment. As it was commented before the horizontal error for the best for navigation, best and nearest 10 m accuracies are very similar. The best minimum value among them is 45 cm surprisingly for the desired accuracy nearest 10 m while the

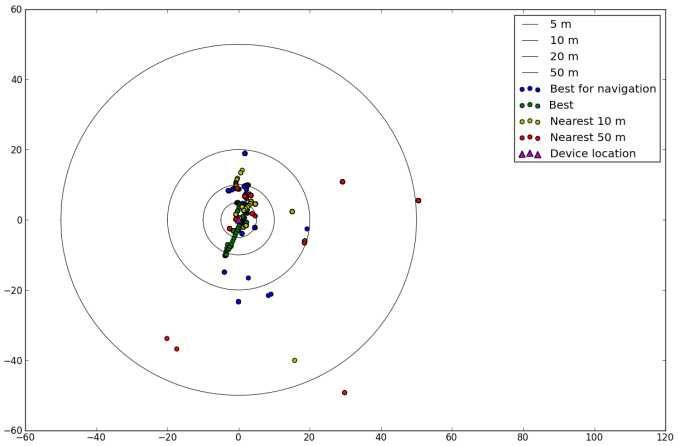


Figure 2.9: Scatter plot of horizontal accuracy for different desired accuracy in iOS in meters

maximum is for the same desired accuracy and it is 43 m. As for the mean the best mean error was performed by the desired accuracy nearest 10 m and it is 8.7 m, although the values for "best for navigation" and "best" are very close. The RMSE values are for the three accuracies 11. .

The better results for desired accuracy nearest 10m instead of for best for navigation or best which are supposed to be more accuracy could be explained using the way that the test were performed. In each location the test for each accuracy were performed one after the other starting with best for navigation continuing with best, nearest 10 m and nearest 50 m. As Apple states in its official iOS 5 website, the GPS accuracy improves during time since it can take several minutes to locate all the visible satellites[22]. Therefore, this can be the explanation of why the results are better in best than in best for navigation. The results for the desired accuracy nearest 50 m are slightly worse than for the other three as it was expected. The horizontal error average is 17.3 m, which is more than twice the one obtained for nearest ten meters.

Zandbergen did a similar study with an iPhone 3G which includes the same A-GPS chip, it can be seen in [Zan09]. He obtained a horizontal error mean of

[22]iOS 5: Understanding Location Services `http://support.apple.com/kb/HT4995`

Table 2.2: Horizontal positional accuracy for different desired accuracy in iOS under real conditions in meters

Accuracy	min	max	mean	RMSE
Best for navigation	1.3581	23.3408	9.1048	11.2679
Best	0.5834	19.5479	9.2488	11.6861
Nearest 10 m	0.4516	43.0338	8.7107	11.3551
Nearest 50 m	0.7861	57.5612	17.2975	26.0383

6.9 m, minimum error of 0.4 and maximum of 18.5 m. The mean error is 1.8 m smaller that the one that has been obtained in this study. This can likely be attributed to two different reasons. The main one is that in Zandbergen's study the iPhone was placed in an ideal position in a tripod while in this study the iPhone was placed in a jeans pocket. This produces that clothes attenuate the GPS signal that the iPhone receives in our experiment. The second reason it is that the Zandbergen's experiments last for 20 min while our lasts for four times two minutes. The GPS improves the accuracy during the time, thus the accuracy in larger experiments has to be better. Apple states in its documentation[10], that the horizontal accuracy is often better than 10 m which is proved in this analysis.

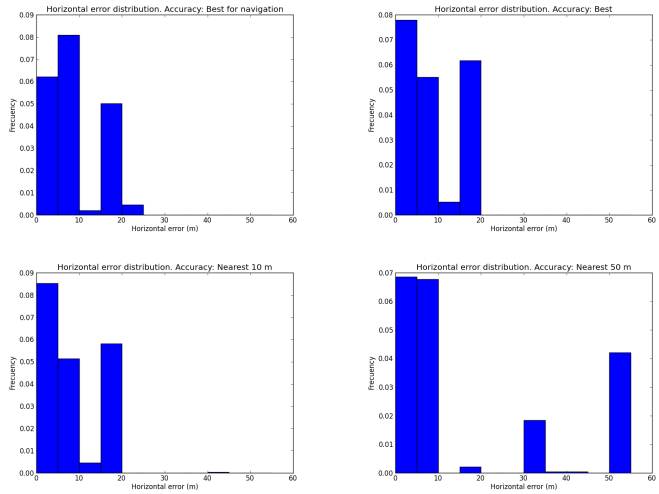


Figure 2.10: Horizontal error distribution for the different desired accuracy in iOS

Figure 2.10 shows the distribution of the horizontal error for the different desired

accuracies. As can be seen the distribution for the desired accuracies best for navigation, best and nearest 10 m is very similar. It can be said that it is almost the same. The horizontal errors from 0 to 10 are highly probable while from 10 to 15 are not very common. From 15 to 20 are likely again and bigger than 20 m are very unlikely. As for nearest 50 m, the most common error is between 0 and 10 but there are errors bigger than 10 that are probable as well, such as from 30 to 35 or from 50 to 55. The same data show in other way can be seen in Table 2.3. As can be seen using the three most accurate options the probability that the horizontal accuracy for a received location is smaller than 20 m is almost 1. As for a horizontal accuracy of 10 m is almost 0.7.

Table 2.3: Percent of locations whose horizontal accuracy is smaller than a certain threshold (shown in each column)

Accuracy	<5	<10	<20	<50
Best for navigation	31.11%	71.61%	97.70%	100.00%
Best	38.95%	66.52%	100.00%	100.00%
Nearest 10 m	42.62%	68.35%	99.79%	100.00%
Nearest 50 m	34.17%	67.92%	68.97%	78.62%

The battery consumption was supposed to be included in the study. Apple provides a tool which is called Instruments[23] to do this kind of analysis. This program has a template, which is called energy diagnostic that provides a diagnostic regarding energy usage. This template was used to store the information about the battery consumption during the different tests. The energy usage level shows the relative energy usage on a scale of 0 - 20. Not big differences were found executing the app with the different desired accuracies. In all the cases this parameter varies from 13 to 18 while the GPS was enabled. There was almost no difference during the periods of time that the GPS was enabled and disabled as can be seen in Figure 2.11. This is not a correct result since the GPS is said to be one of the most power consuming hardware in the smartphones. For this reason, this parameter was considered not accurate enough to be included in the analysis.

The conclusion of this section is that the difference between the three most accurate *desired Accuracy parameters* is insignificant. Best for navigation, best and nearest 10 m provides very similar results in the entire tests, thus the selection of one or another in the final app is not going to change the results. But Apple does not recommend to use the parameter best for navigation unless the device is plugged in[24]. Therefore, best for navigation cannot be used for

[23]Instruments User Guide `http://developer.apple.com/library/ios/documentation/DeveloperTools/Conceptual/InstrumentsUserGuide/InstrumentsUserGuide.pdf`

[24]Core Location Constants Reference `http://developer.apple.com/library/ios/#DOCUMENTATION/CoreLocation/Reference/CoreLocationConstantsRef/Reference/reference.html`

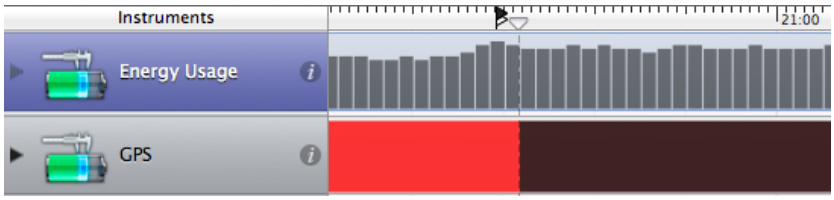


Figure 2.11: Instruments screen shoot which shows battery and GPS probes. Red or black color bar shows when the GPS is enabled or disabled respectively

the final system. Although the difference between nearest 10 m and best were so small, best is going to be used as the desired accuracy parameter. Since the GPS is not accurate enough for the high accuracy tracking that is the aim of this application, the best accuracy that the iPhone can provide has to be used. Moreover in the battery consumption analysis was shown that the difference in the battery consumption between the different parameters was insignificant.

2.4.3 Observed versus estimated accuracy in iPhone

One of the most important parameters that Location Manager provides is *horizontalAccuracy*; this parameter provides an estimation of how accurate the provided location is. As it has been observed in the testings, this parameter can vary from 15 m to 149000 m. Although the normal values that have been observed in the DTU testing are usually below 100 m after few seconds after the location manager is started. But one question that may arise when we are speaking about estimated accuracy is how accurate this estimation is. This is what it is going to be analyzed in this section.

The data set used in this section is the same one that was used in the previous one. The error calculated in the previous section was used to plot it in relation with the horizontalAccuracy that the location manager provides for each location. Apple does not provide any information about how this horizontalAccuracy is calculated but in conventional GPS devices this parameter is calculated using some algorithm derived from Dilution of Precision (DOP)[25], which is based on the number of satellites that have been used to calculate the location. A scatter plot with the data can be seen in Figure 2.12.

[25]Dilution of Precision `http://www.nrem.iastate.edu/class/assets/nrem446_546/week3/Dilution_of_Precision.pdf`

The first two conclusions that arise while seeing the graphs are that the predicted horizontalAccuracy is always bigger than the real one and the correlation between the predicted error and the real one is very weak, same results were obtained in [ZB11]. These two facts make the provided location even less accurate since the point of view of the developer. The location is known not to be very accurate but if the horizontalAccuracy would be accurate, the position could be somehow corrected. In this case, the comparison between two positions could be useful. But neither the location nor the estimated error are accurate, this is the worst possible scenario.

Looking the Figure 2.12 carefully, it can be seen that although the correlation between the two variables is very low, higher estimated error are likelier to have higher real errors. Other relevant fact is that the estimated error tends to have primarily three values that are 17.07, 47.42 and 76.36 m. Moreover, there is not correlation between the different desired accuracies values and the estimated error as it was commented in the previous section. Therefore, histograms have been plotted for all the desired accuracies together for the three more common estimated errors.

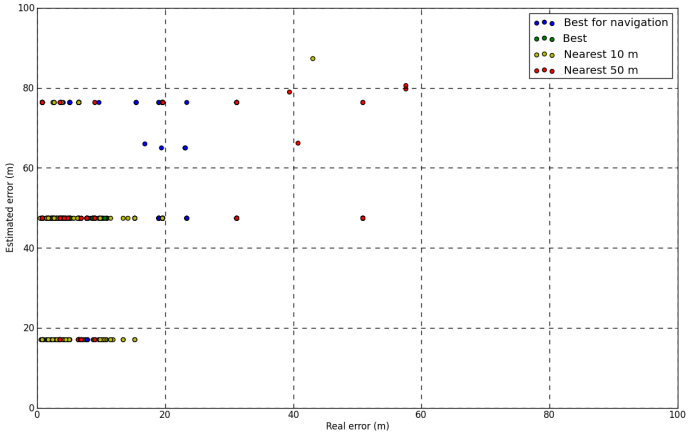


Figure 2.12: Comparison between estimated (by the location services) and real horizontal error

These histograms can be seen in Figure 2.13, it can be observed that there is some correlation between the estimated and the real error. But the difference between 17 and 47 m are very small. As can be seen the only difference is that a real error between 15 and 20 m for 47 m is likelier than in the case of 17 m,

but in both the most probable real error is between 0 and 10 m.

As it has been exposed the estimated error is not that accurate as it should be for values smaller than 60 m. But it can be useful for filtering out the positions that have an estimated error very high and that are almost always very inaccurate. These positions are usually the ones that are obtained using cellular positioning. The accuracy of these positions is enough for many kinds of applications that only needs to know in which are the user is but they are not accurate enough for high accuracy tracking.

The conclusion of this section is that although the correlation between the estimated and the real error is low, it exists. Moreover, the estimated error can be very useful to discard the locations that are very inaccurate because they usually have a high estimated error, higher than 100m.

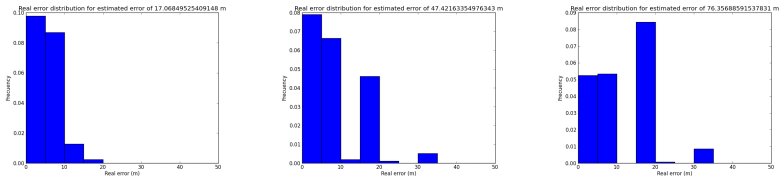


Figure 2.13: Real horizontal error distribution for the different estimated horizontal errors

2.4.4 Time to first-fix in iPhone

Apart from the accuracy, the time that the location services needs to provide an accurate position is a very important parameter. This is usually measured with the time to first fix (TTFF), which is the time that a GPS receiver needs to search for the available satellites and provides the current position. TTFF is usually classified into three different start types, which are cold, warm and hot. These parameters are not defined in any standard; the different GPS manufacturers provide their own definitions [ML08].

For this analysis only cold start is under consideration because Apple does not provide any information about how the downloaded location information for the hybrid system is kept in the internal memory. In April 2011, two researchers found that iPhone stored all the WiFi networks and cell towers around you in an unencrypted database in the internal memory[26]. After that, Apple published

[26]Tracking File Found in iPhones `http://www.nytimes.com/2011/04/21/business/21data.h`

a press release answering several questions about it. In this press release, it is said that after the next software update the cache with the location data would be removed each time the location services are switched off[27]. Therefore, for this analysis it was assumed that each time the location services are switched off, all the cached data used for it is removed from the internal memory of the device. Before each test the location services were disabled and enabled.

It is worth noting that the results of this test strongly depend on the speed of the available Internet connection since the data used for the location services is mainly downloaded from the Internet. In every test the Wi-Fi was enabled and the device was connected to the DTU eduroam network in many of them, otherwise the device had 3G connection.

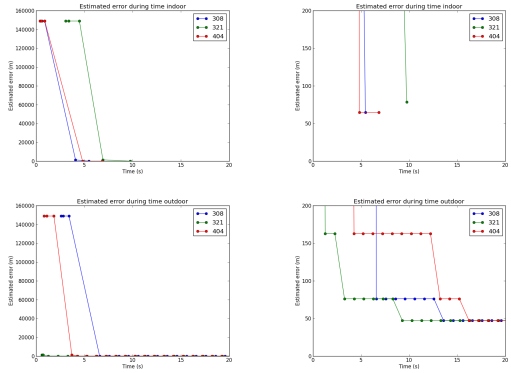


Figure 2.14: Estimated horizontal error during the time for the different indoor and outdoor test

The Figure 2.14 shows how the estimated accuracy varies with the time for the three different outdoor and indoor locations. As can be seen for five out of six test the first location had an accuracy of 149000 m. This first location is certainly obtained using cellular positioning. After this the locations that are received improved with the time and for the entire tests a location with a horizontal accuracy better than 100 m was obtained in less than 15 seconds, which are a pretty good results. Since a conventional GPS receiver needs more than 30 seconds to obtain its first position after a cold start [ML08].

It is obvious in the Figure 2.14 the outdoor and indoor positions have a differ-

tml

[27] Apple Q&A on Location Data http://www.apple.com/pr/library/2011/04/27Apple-Q-A-on-Location-Data.html

Table 2.4: Time in seconds to receive a position with estimated horizontal accuracy better than 100 m in cold start

Location	Time (s) indoor	Time (s) outdoor
308	5.45	6.58
321	9.74	3.27
404	4.81	13.21
mean	6.67	7.69

ent behavior in the range of positions with estimated accuracy smaller than 200 m. While the outdoor positions obtain new positions continuously, the indoor positions obtain an accurate position, which is very likely based on Wi-Fi positioning. Two of these positions have horizontal accuracy equals to 65 which as was commented in this report it was assumed that it was Wi-Fi positioning. As can be seen in the outdoor locations the accuracy improves within the first 20 seconds and in all of them an accuracy of 47 m is achieved.

In the Table 2.4, the time to first fix has been defined as the time that is needed to obtain a location with horizontal accuracy better than 100m. 100m was selected because as it was shown in the previous sections, the majority of the "accurate" positions have an estimated error in this range. As can be seen in the table the mean time to receive one of this locations in a cold start is 6.67 for indoor locations and 7.69 for outdoor ones. The Wi-Fi positioning looks to be around 1 second faster than the A-GPS, although the A-GPS improves the locations in the following seconds after the first fix and Wi-Fi positioning do not do that. The different times for the three different locations have big variations from one to another.

The conclusion of the analysis of this section is that the time to receive a location with a horizontal accuracy of 100 m varies from one experiment to another but it is always better than 15 seconds and in the majority of the tests is better than 10 seconds. Other relevant discover of this section is that indoor positioning does not improve during the time while A-GPS improves during the first 20 seconds. A very relevant aspect is that the iPhone was able to provide a quite accurate location in all the indoor/ outdoor test not only in this analysis but also in all the test that were performed for this master thesis. Thus, the availability of iOS positioning in the DTU area is almost 100%.

2.4.5 Comparison between different devices

All the analysis than have been explained in the previous sections have been performed using an iPhone and static data provided by utopian city_scape. But in this section, the horizontal accuracy of the iPhone is going to be compared with the one that other GPS enabled device provides. This device is Garmin forerunner 405 CX. The analysis is similar than the one performed for the different *desiredAccuracy* parameters in the previous section.

As can be seen in Figure 2.15, the positions obtained using the Garmin watch are usually more accurate that the ones that iPhone provides. These results were expected since the Garmin watch is not a general use device like the iPhone; it is a device which main feature is that contains a GPS sensor. Moreover, Garmin is a company specialized in GPS enabled devices.

The different locations do not follow any clear pattern in any of both devices. Although for the iPhone the locations tend to be in the diagonal between the first and third quadrant, the same was observer in the analysis of the desired accuracies. As it was said in this section, no conclusions can be achieved from this. It is remarkable that all the locations that the Garmin watch provides were inside the circle of 10 m radius. This is a very important fact since it assures that once the location is provided is at least 10 m accurate. In the case of the iPhone unfortunately this does not happen.

Table 2.5: Comparison of horizontal positional accuracy between iPhone 3GS and Garmin forerunner 405 CX in meters

Device	min	max	mean	RMSE
iPhone	0.7970	55.3629	6.9375	8.8528
Gramin watch	1.4947	8.9773	5.263	5.8425

The Table 2.5 shows the numeric results. As can be seen the mean horizontal error is 6.93 m and 5.26 m for the iPhone and the Garmin watch respectively. The Garmin mean error is better than the iPhone one mean in all the studied cases. Although the horizontal error presented by the iPhone in this analysis is almost 2 m smaller than the one presented in the Section 2.4.2. After a analysis of the experimental data, it was discovered that for one of the location, iPhone was able to provide a position with an accuracy of 0.79 m during a long period of time. This produces that the mean of the horizontal error decreases a lot since just six points were analyzed in this particular study.

The most remarkable result of this table is that the positions that the watch obtains are in a smaller range around the real position that the ones that the

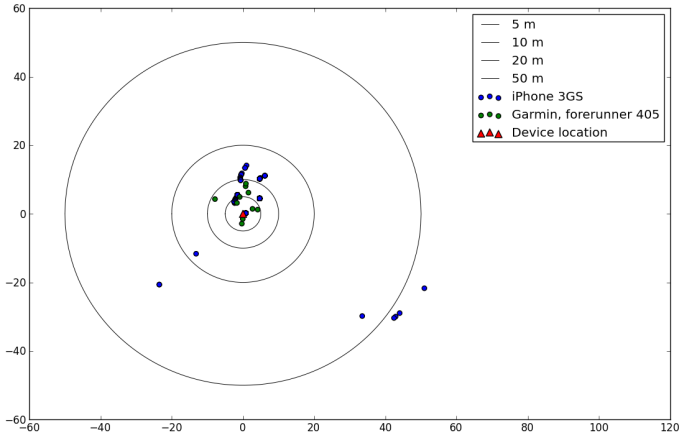


Figure 2.15: Scatter plot of horizontal accuracy for iPhone and Garmin forerunner 405 in meters

iPhone provides. This can be seen observing the iPhone RMSE value which is bigger than Garmin one. Moreover, the max and min values are much spreader for the iPhone than for the Garmin that its maximum error is 8.9 m. Remember that this was the best mean error for the iPhone in the study of the Section 2.4.2.

The conclusion of this section is that as expected the Garmin watch is at least 2 meters more accurate than the iPhone in static positions. Moreover it produces all the locations within 9 meters distance from the real location.

2.5 Analysis of Measurements in motion

This section analyzes the accuracy that the iPhone 3GS provides tracking real routes at the DTU campus. To be able to measure this error, the Garmin forerunner 405 CX was used as the reference path. The real accuracy, estimated accuracy, time between received positions and time between different positions are analyzed to try to find repetitive patterns when the user is entering a building or the GPS accuracy decreases drastically. Detecting correctly this action is crucial to obtain a realistic representation of the tracked trajectory in the model.

2.5.1 How to measure the real route

The first problem that arises when the error in a given route has to be calculated is how the real route can be measured. This is a very big problem because the user not only has to remember what the route was, but also when he was in each position. This second fact makes almost impossible to obtain manually an accurate measure of the routes with the means that were available for this master thesis.

One first approach could be to leave the time parameter out of the analysis, as it was done in [ZB11]. But analyzing the data that the iPhone provides, it was noticed that when using GPS positioning the same position tends to be received many times. Thus, it can be though that the iPhone does not always provide your current location, it sometimes provides old locations during a certain period of time, this fact is analyzed in the following sections. Therefore the time parameter can not be left out of the analysis.

If the time parameter cannot be left out of the analysis, a GPS enabled device more accurate than the iPhone has to be used to obtain the real route. This solution provides the better balance between accuracy and means needed to make the analysis. The device used to measure the reference route was the Garmin forerunner 405 CX. This device was selected because it was available to use and provides the possibility of importing the routes into a computer using a GPX file.

The Garmin watch was found to be less than 2 meters more accurate than the iPhone in positional positions, in the Section 2.4.5. But as can be seen in the route‘s maps that are shown in the Appendix B, the watch is way more accurate in measurements in motion. It provides positions more often and these positions are more accurate than the iPhone ones. This can be seen especially when there are some curves in the route. In these cases, the iPhone does not show the exact

curve while the watch usually provides enough positions to be able to see the whole curve.

However, it can not be assumed that the watch is perfect, it will have some error. Therefore, the horizontal error calculated in this section can not be assumed as a real accurate error calculation as it was for the fixed position. It is actually an estimation of the horizontal error that the iPhone makes. It can be very useful to compare between different estimations of the same route in which some positions have been removed to obtain a more accurate route[28].

2.5.2 GPS data processing

The data processing in this section is divided into two main parts, the first part consist of joining the data from the iPhone and the watch to plot it in a map and the second one is the numerical error calculation. These two different tasks were done using geographic coordinates and UTM coordinates respectively. The data was harvested in the same way that was done for the static positions analysis explained in Section 2.4.1.

As for the routes map creation, the Google static maps API[29] was used. This API provides the possibility of creating map images with marker and paths. The map is generated using URL parameters that are sent using a standard HTTP request; the map is received as an image. The main limitation that this API has is that the URL has a limited size, which is 2000 characters. For the large routes that are analyzed in this study much more than 2000 characters are needed. The solution for this problem is encoding the routes using encoded polylines[30]. Since the program that generates de map was programmed in python a python polyline encoder was needed. The used python encoder was py-gpolyencode[31].

As it was explained before, the error that the iPhone provides is going to be calculated in relation to the route that the Garmin watch obtained. It can be many ways to measure this distance. For this analysis, the error between each unique iPhone position and the position at that time in the path that the Garmin produced is the error that is going to be measured. Therefore, each unique iPhone position will have a calculated error. Instead of measuring the minimum distance between each given iPhone position and the Garmin path,

[28] Calculate distance, bearing and more between two latitude/longitude points `http://www.ig.utexas.edu/outreach/googleearth/latlong.html#ellipsoid`

[29] Static Maps API V2 Developer Guide `https://developers.google.com/maps/documentation/staticmaps/`

[30] Encoded Polyline Algorithm Format `https://developers.google.com/maps/documentation/utilities/polylinealgorithm`

[31] py-gpolyencode `http://code.google.com/p/py-gpolyencode/`

the time that the given iPhone position was received is used to calculate the point at that time in the Garmin path. For this calculation, it has been assumed that the movement between two Garmin positions was constant during the time and in a straight line. These calculations are made using UTM coordinates. Thus, the horizontal error is just the Euclidean distance in meters between the two coordinates.

To try to obtain both routes synchronized, the iPhone and Garmin watch were started to obtain locations at the same. The Garmin watch provides in the GPX file when the route was started and when each location was received. Thus it was pretty easy to synchronize both. Although as can be seen in the final results this synchronization is not perfect and will introduce some error. That is why the calculated error has to be seen as a relative error that can be useful to compare between different point selections within the same route. It can not be seen as a real accurate error, which can be used to compare between different routes.

Not only the maps were plotted, but also some graphs showing the error and the time for the different routes. The most important graphs can be seen in the Appendix B. These graphs were plotted using the python library Matplotlib [32].

The locations that the iPhone provides were filtered out following several criteria that will be explained in the following sections. All the calculations were done for all this set of points. Moreover, only the unique positions were considered for this analysis. Since the parameter distanceFilter was set to zero in the entire test, the location manager provides the same position many times after it provides a new one. The time rate that this positions are received is going to be analyzed as well. But, for the error calculation the repeated positions are ignored. Two positions that are equal but with a different predicted horizontal accuracy are considered as different positions.

2.5.3 Calculated horizontal error

As can be seen in Appendix B, nine routes were performed in the DTU area during the months of February and March 2012 for the development of this master thesis. In this section, the total error calculated for these routes are going to be presented. This section shows the main results for all the routes, although the results for each of the nine routes in detail can be seen in Appendix B. The analysis performed in this section was done in the same way as it was done for the static locations. The calculated horizontal error is studied with the

[32] http://matplotlib.sourceforge.net/

scatter plots, numerical values of the distribution and the histograms.

The scatter plot can be seen in 2.16. As can be seen in the scatter plot, the location of the different positions does not follow any pattern. It can be seen that the points tend to be placed in the diagonal between the first and the third quadrant more often than other places. But this tendency is very weak, thus it will be ignored from now on in this master thesis. Same tendency was found for the static positions, but in this case the pattern was more obvious. Other conclusion that can be obtained from the scatter plot is that the correlation between the predicted and the real error is very weak. The same conclusion was obtained in the case of the static positions.

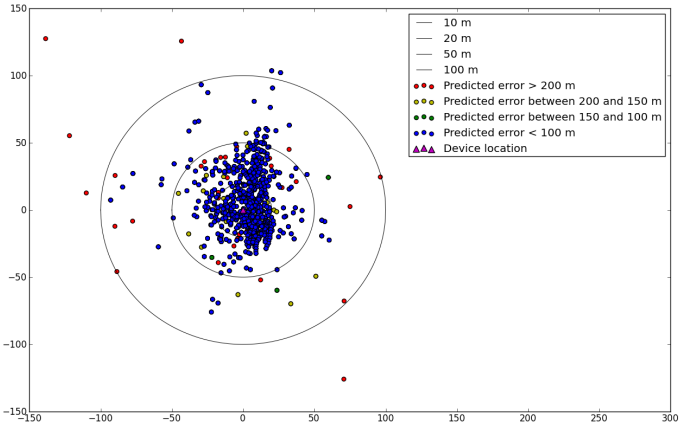


Figure 2.16: Scatter plot of horizontal accuracy for all the routes

The results shown in Table 2.7 were obtained using all the unique positions that the iPhone obtained but the ones that have a predicted accuracy higher than 1000 m. The location manager usually provides these locations when the location service is started and are sometimes outside DTU and far away from the real location, thus it does not make any sense to include it in the analysis.

The nine routes can be divided in three types of routes: routes that go inside a building, outdoor routes with curves and straight routes. The four first routes that appear in the table belong to the first group, routes which go inside a building. These routes are from 308 to 342, from 341 to 101, from 343 to 308 and from library to 229. The second group is outer routes that are from 115

Table 2.6: Horizontal accuracy for the different routes in meters

Route	Min	Max	Mean	RMSE
From 308 to 342	7.46	188.34	26.59	38.80
From 341 to 101	2.28	175.08	30.86	40.05
From 343 to 308	1.04	133.87	36.23	44.07
From library to 229	6.13	132.91	35.49	47.02
From 115 to library	4.24	99.34	24.94	30.97
From 229 to 115	3.24	61.52	26.33	29.92
From Nordvej to 302 biking	4.62	93.08	26.57	33.60
From Knuth-Wintherfeldts to Nils Koppels Alle walking	3.76	41.47	17.83	19.02
From Nils Koppels Alle to Knuth-Wintherfeldts biking	4.69	218.90	30.32	38.33
All	1.04	218.90	27.13	35.10

to library, from 229 to 115 and from Nordvej to 302 biking. The last group is straight routes that is composed of two routes From Knuth-Wintherfeldts to Nils Koppels Alle walking and From Nils Koppels Alle to Knuth-Wintherfeldts biking.

Analyzing the mean errors for these three different groups of routes in the Table 2.7, it can be observed that have different mean values. The mean value for the routes that cross a building is always bigger than the outdoor routes one. This result was expected since when the mobile go inside a building the GPS signal are lost and it takes a while to detect it again once it is outside. During this period without GPS signal either Wi-Fi positioning or inaccurate GPS positions are received, this produce degradation in the mean error. The straight routes should have better mean error than the outdoor with curves. This is true in the case of the route for Knuth-Wintherfeldts to Nils Koppels Alle that has the lowest mean error between all the studied routes. But the route from Nils Koppels Alle to Knuth-Wintherfeldts has a mean error, which is quite high. This has not a direct explanation but the time between unique positions in this route is the fastest between all the analyzed, one second. And the horizontal error has a high value and consecutive a low value all the time, thus it can be think that the iPhone does not provide very accurate timestamp of the locations with this time rate.

It can be observed that the mean error for all the routes is 27.13 m, which is much higher than the one obtained for the static locations in the Section 2.4.2. Just looking the maps it can be seen that the error is higher than in the static positions since the rate which the locations are received is not high enough for provide an accurate route and the positions that are received are not accurate neither. However, the real error should be smaller than this because as it was explained before the route reference is not the real one, it is an estimation.

As can be seen in the Figure 2.17, the distribution of the horizontal error, the error between 10 and 20 meters are the most likely to happen. Errors higher than 20 meters are less probably as the error value increases. Moreover, error smaller than 10 meters are less probable as the error value decreases.

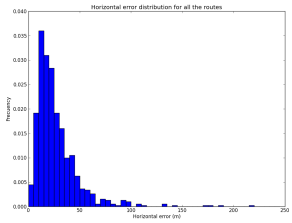


Figure 2.17: Horizontal error distribution for all the routes

The conclusion of this section is that the iPhone has a bigger horizontal error for measurements in motion than for static ones. Moreover, the straight routes have a smaller error than the ones that makes curves. The routes that combine curves and crossing building are the ones that have a bigger horizontal error. As the histogram shows the most likely horizontal errors are between 10 and 20 m.

2.5.4 Filtering out inaccurate locations

As it was analyzed in Section 2.4.3, the predicted horizontal error that the location manager provides is not really accurate for small values (smaller than 100m). But it can be very useful to know if the position is really inaccurate and if the position is based in GPS positioning, Wi-Fi positioning or Cellular positioning.

As can be seen in the Appendix B in the red routes, which includes all the positions that the iPhone obtained. There are parts of the routes where the iPhone provides pretty accurate positions, and there are other parts especially at the beginning and when a building is crossed that the positions are very inaccurate. These positions can have error of several hundred meters, which is inacceptable for high accuracy tracking applications which is the aim of this master thesis.

Two examples of these very inaccurate positions can be seen in Figure 2.18. In all the route maps in this master thesis the blue line is the route that the Garmin Watch provided, the red one is the route that the iPhone obtained using all the

Figure 2.18: Two examples of very inaccurate positions (inside yellow circle) in routes: from 343 to 308 (left) and from Nils Koppels Alle to Knuth-Wintherfeldts (right)

positions and the green one is the one that is obtained after applying horizontal error and Wi-Fi positioning filter. The points that have high horizontal accuracy values are marked with a yellow circle. These types of positions have to be filtered out in order to obtain a more accurate tracking since it does not provide any relevant information. Analyzing the error graphs that are available in the Appendix B it was discovered that all this positions had a very high predicted horizontal accuracy. It was usually higher than 700 m and always higher than 100m. This positions are obtained using GPS but with few satellites available or using cellular positioning. As it was discovered during the development of this report, the predicted accuracy of the positions that are obtained using cellular positioning is always high, usually higher than 1000 m, and this predicted accuracy is always an integer instead of a float number. The predicted accuracy of the two examples that are shown in Figure 2.18 are floats number, thus it can be guessed that these positions are obtained using satellite positioning but with weak signal strength.

As can be seen in Table 2.7, just removing the positions with accuracy higher than 100 m the results are way better especially in the routes that the results were very inaccurate. But, it was observed that sometimes when the GPS positions are not very accurate the location services switches to Wi-Fi positioning when you are outside. This fact worsens the tracking since these Wi-Fi positions are usually more than 50 m far from the real position. In the Figure 2.19, it can be seen two part of two of the analyzed routes with this Wi-Fi positions removed on the right or not removed on the left.

Removing some of the positions obtained using Wi-Fi positioning was found to

Table 2.7: Horizontal accuracy for all the routes with different error and Wi-Fi filters in meters

Error filter	Wi-Fi filter	Min	Max	Mean	RMSE
1000	0	1.04	218.90	27.13	35.10
200	0	1.04	105.53	24.85	29.67
150	0	1.04	105.53	24.35	29.09
100	0	1.04	105.53	24.21	28.91
100	2	1.04	105.53	24.19	28.90
100	3	1.04	105.53	24.19	28.90
100	4	1.04	105.53	23.91	28.38
100	5	1.04	105.53	23.91	28.38
100	6	1.04	105.53	23.99	28.46
100	7	2.35	105.53	23.98	28.48

be a very good idea, but some of them are very useful for example to detect when the user is inside a building. Moreover, in the case of the device is inside a building the Wi-Fi positioning positions were found to be the most accurate available. To try to find the perfect valance between these two situations, a Wi-Fi positioning filter was defined as the minimum number of Wi-Fi positions that have to be provided in a row to use it. For instance, if the filter is 4 only the Wi-Fi positioning that is received in groups of four or more will be used and the ones that are received in groups of three or less will be removed and not used for the route calculations. As in a row it refers to that not other positions but Wi-Fi positioning ones are received between them. The Wi-Fi positioning filter has to be applied after the horizontal error positioning, since the location services sometimes provides positions with accuracy higher than 100 m when the user is inside a building. As was commented before in this report the Wi-Fi positions can be differentiated because the predicted horizontal error is equal to 65 m in almost all of them.

Several Wi-Fi positioning filters were analyzed to try to find the best number for this parameter using all the routes. The results can be seen in Table 2.7. As can be seen in the table, the best result were obtained for error filter of 100 m and Wi-Fi filter of either 4 or 5. The mean error was improved almost 4 meters which is an awesome result since only two easy of implement filters have to be used to obtain these results. Error filters smaller than 100 are not considered since as it was seen in Section 2.4.3 the correlation between predicted error smaller than 100 m and the real error is very weak, thus positions with predicted error 76 can be more accurate than other positions with smaller predicted error.

The green route in the maps that are shown in the Appendix B were obtained using an error filter of 100 m and a Wi-Fi filter of 4 meters. As can be seen the routes are much more accurate that the ones that the iPhone directly provides.

Figure 2.19: Two examples of removing Wi-Fi positions (inside yellow circle) in routes: from 341 to 101 (above) and from 343 to 308 (below)

2.5.5 Time and speed between different locations

In this section, the frequency which new coordinates are received is going to be analyzed. The routes used for this section have been filtered with an horizontal error filter of 100 m and Wi-Fi filter of 4 samples, since they are the filters that provide better results as it was seen in the previous section. Moreover, all the repeated positions have been filtered, thus the routes only contains unique positions for this section. As was commented in previous sections, the iOS location manager usually provides the same coordinates many times and this fact is undesired for a time analysis like this. In these cases, the time for the first position, which includes this coordinates, have been used as the timestamp for this position. Sometimes, the location manager does not change the coordinates but change the predicted accuracy. In case of better predicted accuracy, this is

Table 2.8: Time between unique positions for all the routes in seconds

Route Name	Min	Max	Mean	RMSE
From 308 to 342	0.06	72.56	16.23	26.59
From 341 to 101	0.50	104.11	7.89	19.25
From 343 to 308	0.99	140.00	23.90	39.77
From library to 229	1.62	104.85	32.83	45.96
From 115 to library	7.00	64.00	23.44	29.34
From 229 to 115	1.58	61.96	29.83	34.06
From Nordvej to 302 biking	0.99	30.00	4.61	7.19
From Knuth-Wintherfeldts to Nils Koppels Alle walking	0.93	51.65	3.47	7.47
From Nils Koppels Alle to Knuth-Wintherfeldts biking	0.97	38.00	1.63	3.79
Total	0.06	140.00	6.50	15.80

updated but not the timestamp.

The time when each location is received is important for two main reasons. The first one is that it can be used to calculate the speed at which the user was moving; this speed should be used to move the users in the model correctly. The other reason is that it is useful to know the period of time between locations to design the route-improving algorithm correctly, this is explained in the next chapter.

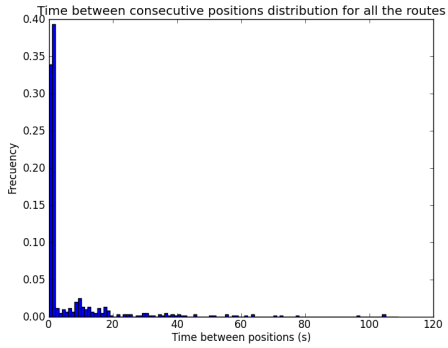


Figure 2.20: Time between position distribution for all the routes

As can be seen in the Figure 2.20, the most likely times between new positions are 1 or 2 seconds. Many new positions were received in a period of time between 2 and 20 seconds. It is not very likely more than 20 seconds between new positions but as can be seen in the distribution it can happen. This has to

Table 2.9: Speed between positions for all the routes in kilometer per hour

Route Name	Min	Max	Mean	RMSE
From 308 to 342	0.48	245.17	16.68	50.21
From 341 to 101	0.02	45.03	18.06	20.65
From 343 to 308	0.88	91.34	10.68	23.08
From library to 229	0.64	70.15	13.17	22.80
From 115 to library	2.67	9.80	5.98	6.38
From 229 to 115	2.78	43.24	10.10	14.72
From Nordvej to 302 biking	0.77	37.96	15.96	17.59
From Knuth-Wintherfeldts to Nils Koppels Alle walking	0.99	13.88	6.54	6.93
From Nils Koppels Alle to Knuth-Wintherfeldts biking	1.72	37.63	18.95	19.77
Total	0.02	245.17	13.66	19.11

Table 2.10: Absolute speed for all the routes calculated as the total time used to make the whole route in kilometer per hour

Route Name	Absolute Speed
From 308 to 342	4.18
From 341 to 101	6.31
From 343 to 308	4.22
From library to 229	4.28
From 115 to library	4.70
From 229 to 115	6.21
From Nordvej to 302 biking	12.82
From Knuth-Wintherfeldts to Nils Koppels Alle walking	6.17
From Nils Koppels Alle to Knuth-Wintherfeldts biking	15.20

be taken into account in the development of the algorithm that represents the users in the model.

The Table 2.8 shows the same data that was represented in the graph but for every route. As can be seen, the mean time including all the routes is 6.50 seconds, which is not really high, but for many individual routes this mean time is higher, and the maximum time was 140 seconds, which is really high. If we think in terms of the route groups that were explained before we can see that the time between positions have some correlation between routes that are from the same type.

The routes that go inside a building have the highest maximum value, which makes sense since when the user goes inside the GPS signals are lost and the location manager starts to provide inaccurate positions which are filtered out in our system. Moreover, it can be observed that the routes that are straight or by bike have a mean time way smaller than the rest. The conclusion that we can obtain from this is that the GPS works better in these situations.

Not only the time is needed to calculate the speed but also the distance between the coordinates. Using both the speed between each position has been calculated, the results can be seen in Table 2.9. Taking into account that the routes were made approximately in a constant speed, walking or biking, it can be stated that the speeds that it is shown in Table 2.9 are really inaccurate. The means have no correlation between the different routes and the min and max speeds have a big difference. For this reasons the instant speed between two locations can not be used to represent the users in the model.

Since the values for the instant speed are not accurate the absolute speed for each route has been calculated and can be seen in Table 2.10. These results are pretty accurate and the difference between the walking and the biking routes can be observed.

The conclusion of this section is that the timestamp that the location manager provides for each location is not precise. Therefore, it can not be used to calculate the instant speed between two locations. For the representation of the users in the model the speed should be calculated for a group of several locations instead of between each two.

2.5.6 Trying to detect when the user goes inside a building

In order to represent the users in the model in a more realistic way, it would be really useful to be able to detect when the user is inside a building. As it was commented in the previous sections if a position is obtained using Wi-Fi positioning the horizontal error that location manager provides is 65. If the user is inside a building accurate GPS positions cannot be obtained, therefore the location manager should use Wi-Fi positioning since it is the most accurate way the location can be obtained. It can be thought that if the received position is obtained using Wi-Fi positioning, the user is inside a building. But analyzing the routes this was found not to be truth because sometimes the location manager provides Wi-Fi positioning locations when the user is outside because the GPS ones are not accurate enough.

After applying the Wi-Fi filter that was explained in Section 2.5.4 with a value of 4, all these Wi-Fi positioning positions that were obtained when the user was outdoor are filtered out. Only the Wi-Fi positions that were obtained when the user was inside pass the filter. They are always in groups of 4 or more positions because if there were positions between them the horizontal error filter would filter it. Because they have a high-predicted horizontal error due to the unavailability of reliable GPS signals.

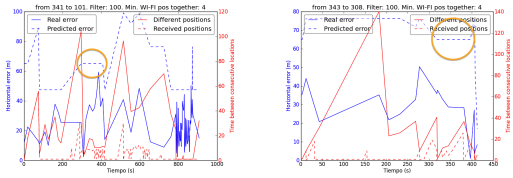


Figure 2.21: Graph of two routes where it was detected that the user was in an indoor location (yellow circle)

This phenomenon can be observed in Figure 2.21, the yellow circles indicates the periods of time that the user has been detected to be inside a building. Looking the map can be seen that these estimations are correct. This algorithm detects when the user goes inside a building in the three routes that the user goes inside a building, which is a really good result.

The detection of when the user goes inside a building can be very useful in the representation of the users in the model. Since the user should enter the buildings using the doors and ones is inside it should stay there without moving. The interior of the buildings is not implemented in the model, and iPhone cannot detect the positions inside the buildings.

Chapter 3

Representing the routes in the 3D model

3.1 Problem description

Unlike it can be thought, representing the routes in the 3D model is not an easy task. As it was studied in Chapter 2, the positions that are received from the smartphones are not accurate enough. They are usually more than 10 m further from the real position and sometimes even more than 50m. These are not the only inaccuracies; the positions do not arrive with constant frequency. Sometimes, a new position is received each second and other times it takes more than 100 seconds to receive a new one.

The purpose of the system that has to be developed as part of this master thesis is to move a character in 3D models in realsite (company website realsite.dk) using positions that a smartphone provides in real time. Realsite uses video games technology to show 3D models of real buildings. The system should move a character following the same rules than when the character is controlled by the user. These mean that the character cannot cross walls, climb a big high... Moreover, the route has to look feasible, this means for instance that the character cannot jump off 5 m. The character has to be placed in the ground all the time to look feasible. It can not be placed in the top a building for example. Therefore, the routes received from the smartphone cannot be

represented in the model directly. They need a transformation to move the character using them. This transformation was developed as part of this master thesis and it is going to be explained during this chapter.

As it was described before, all these problems arise because the location services are not accurate enough. Thus systems like the one presented in this chapter have to be developed if the route is going to be represented in a more sophisticated framework than a plain 2D map. To define the problem more precisely, a set of 9 routes, studied in Chapter 2 were analyzed to establish the different problems that need to be solved with this route representation algorithm. The main problems found are explained in the following subsections.

3.1.1 Coordinates conversion

The first problem that arose when we were trying to represent the coordinates sent by the phone in the model was that the location service in iOS provides the locations using geographic coordinates. The DTU 3D model has its own reference system which center is in the center of the campus. It is obvious that a program that can convert from geographic to DTU coordinates was necessary. And this program was not available, therefore it was developed as part of this master thesis. The program had to be developed in Java, since it has to be executed in Utopian city_scape server, which is programmed in this language. It is necessary to be able to convert from geographic coordinates to UTM coordinates in the server side for this application.

3.1.2 Crossing buildings and points in the top of the buildings

Figure 3.1: The yellow circles indicate different examples of routes crossing buildings and points in the top of the buildings,

These are two different problems but they are very related. These two problems are very common in the analyzed routes; almost every route has these problems. The crossing building problem happens when the path that joins two positions crosses a building. This problem has to be fixed, otherwise the character will hit the building and sometimes it gets stuck in this position. Both are not desired behaviors. To avoid that, some points have to be added to the route to avoid the building without modifying the final route a lot.

Other problem that usually happens at the same time than the previous one is that many positions are placed in the top of the buildings. As it was commented before, the character cannot be placed in the roof of a building; it has to be placed in the ground outside of any building. Therefore the positions that are in the top of a building has to be removed or replaced for other ones from the final route. Both problems can be seen in any of the figures in Figure 3.1.

3.1.3 Changes of altitude along the route

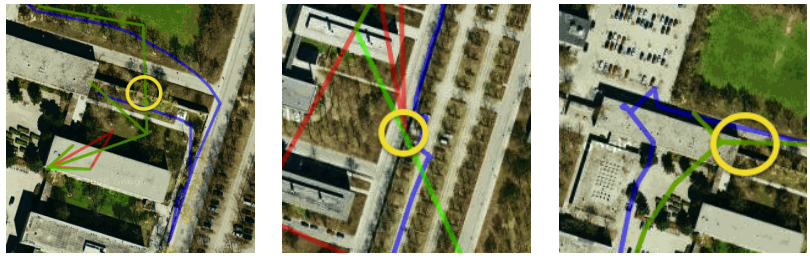


Figure 3.2: The yellow circles indicate different examples of routes changing the altitude

This problem happens when two consecutive points are at different altitudes. DTU campus is an uneven piece of land; it has several high walls that separate this altitude difference. These walls have usually stairs to allow pedestrians to cross it. Since the routes are not very precise, although the user that was being tracked changed the altitude using stairs, the route does not follow that path. This makes the character hit the wall and gets stuck there or jumps off the altitude different. Both behaviors make the route to look unfeasible. This problem has to be fixed adding the corresponding points to change the altitude properly. Three examples of this problem can be seen in Figure 3.2 .

3.1.4 Fast change of heading

As it was observed in Section 2.5.5, sometimes the location services provides new positions each second. This behavior that is apparently desirable ended up being undesirable. These positions are usually really close one to another and do not follow a straight line. This fact made the character change the heading very often. The camera was attached at the back of the character thus the camera moves with the character. This fast movement looks really strange in the model. Since these many positions close one to another does not provide more accuracy than more spread out position. This was considered as a problem that needed to be fixed.

3.2 Tools and concepts required

This section introduces some concepts about path finding and map matching, technologies. The tracking problem that is going to be solved is neither a map matching nor a path finding problem. But understanding the techniques that are used in these fields is crucial to understand the solution designed in this master thesis. Since several parts of it are based in techniques that are used in path finding and map matching. Moreover at the end of the section the used development environment is briefly introduced.

3.2.1 Path finding

Path finding refers to the process of finding the shortest route between two points in a digital environment. This technique is mainly used in video games, although it is used in other applications as well. An example of this can be found in [PM09], where path finding is used in the semiconductor industry. The problem that we are trying to solve in this project cannot be solved using path finding algorithms because these algorithms need an especial environment map. The environment maps that are available for this project are the 3D models that UCS produced. These 3D models are very accurate but they can not be explored in the way that typical path finding algorithms such as A* do [PEH68]. An improved A* algorithm which reduces both exploration and time can be found in [YB06]. Recent work in hierarchical path finding can be found in [Mul04] and [Rab00]. These algorithms need the environment represented as a network, which is composed of nodes and edges. This network needs to be created manually for each 3D model.

Although path finding algorithms cannot be used directly in this project, the problem that has to be solved is somehow a path finding problem. One of the objectives for this section is to find an obstacles free path between any two points in the 3D model. This can be seen as a short distance path finding problem. It varies from the classical path finding problem because the character has to move following the GPS positions received from the phone. An example of a system similar to the one developed for this master thesis can be found in [MYW10]. This article describes the implementation of a Pac-Man Game on campus using the locations that are harvested using a smartphone. Unlike our project, the system designed in this article does not make any transformation in the locations that are received from the smartphone.

Other path finding practical implementation can be found in [MCP]. In this article the character is moved using the voice and the exact path is calculated using path finding algorithms. A technique for generating human motion path in 3D environments can be found in [SB00]. Both studies use a grid map for the path finding algorithm. This map has to be generated manually in our case as well. Thus this technique can note be used directly in our project.

3.2.2 Map matching

Map matching is the process of correlating a sequence of user's positions with the road network. It can handle different data types such as point-to-line, line-to-line and polyline to polyline. In this project point-to-line is going to be used. Regarding temporal-response characteristics, map matching can be classified into online and offline map matching. The online methods work with the received data in real time, while the offline ones work after the whole set of data has been collected. One part of the algorithm that has been developed in this project is a map matching algorithm that only works in case the road network is available. This algorithm is explained in Section 3.3.3.3.

There are many studies in this field that solve different problems. Sometimes neither the location nor the network data are accurate, Christopher, David and Alain proposed several map matching offline algorithms that studied this problem using personal navigation assistants [CEW00]. Jagadeesh, Srikanthan and Zhang developed an online algorithm for real-time vehicle location that only uses GPS data [GRJZ04]. Marchal, Hackney and Axhausen designed an efficient algorithm that relies in the same than the two previous one, GPS data and network topology [FM04]. Other map matching approach for travel/activity research needs can be found in [Zho].

Other offline map matching approach can be found in [TG11]. It uses Douglas-

Peucker [DD73] algorithm to obtain key waypoints in the trajectory. This same algorithm was used in this master thesis. As it was commented before in this master thesis, sometimes the location services are not able to give an accurate position during a long period of time. A map matching algorithm was proposed in Seul National University for these cases [YJs05].

Road Reduction Filter (RRF) [GT99] is a method of detecting the road in which a vehicle is traveling along. This method compares road centerlines and trajectories described by the GPS positions. Further development of this algorithm can be found in [GTAH01].

The map matching algorithm that was developed for this master thesis is in 2D because the road network was provided in 2D, moreover the altitude in the GPS positions is very inaccurate. Regarding this reasons in our particular case using a 3D algorithm would not improve the accuracy. But for other applications where the road network is bigger it can improve the accuracy. Using a map matching 3D algorithm implies to have information of the network in 3D. Menglei propose a way of modelling the road network in 3D in [YZ10].

3.2.3 Unity and the development environment

Unity is a development engine that allows developers to create games and other interactive 3D content. Using Unity you can join your 3D models and assets to create interactive 3D content. This part of the master thesis was developed in Unity, since it is the tool that Utopian city_scape uses for the development of the web player in realsite. Therefore, the integration with realsite was easier than using other tools.

The version of unity used for this project was the free version of Unity 3. Unity allows developers to program in C# or JavaScript, all the libraries can be used in both languages. C# was selected for this project, because it is the language that is used in realsite. The model as it is implemented in realsite is too heavy to be run in unity. Each building in DTU 3D model has two different versions, a high quality version and a low quality one. Therefore, a DTU 3D model combining high and low quality buildings was set up manually for the purpose of this master thesis. Some screenshots of this model during the development phase can be seen in Figure 3.3.

To represent the routes in the model some object had to be moved through the route points. This object could have been any game object from unity or some 3D model imported into unity such as a human avatar. For the development phase, a sphere was used as can be seen in Figure 3.3. It was selected for

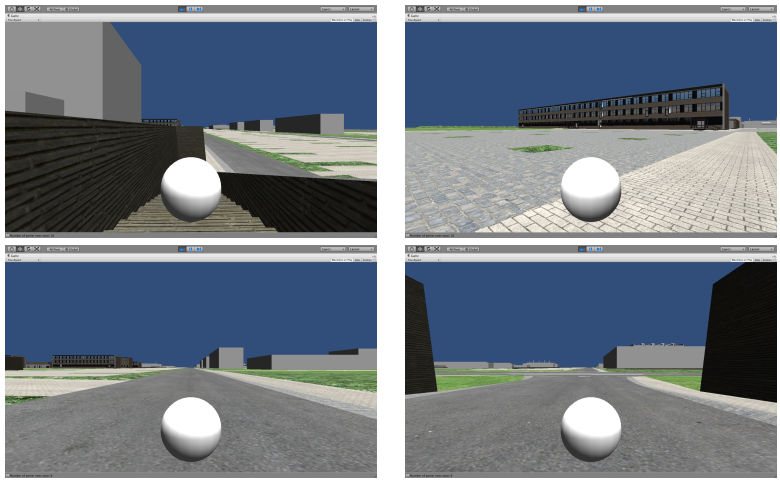


Figure 3.3: Four screenshots of the development phase of the project in Unity

simplicity since it is available directly in Unity. Moreover, once the algorithm it is done this is very easy to change. Each game object in unity can have a script attached to it that controls it. Thus, the ball has a C# script attached to it that is in charge of executing the route improvement algorithm and moving the ball around the DTU 3D model using the route result. The ball has a camera attached at the back of it that is moving with the ball. With this system, the user can see how the ball moves in the model following the improved route.

As it was commented in Section 3.2.1, the problem that it is trying to be solved cannot be solved using path finding algorithms because of the unavailability of an environment map. But Unity has some path finding tools that allow the developer to build a navigation map, which is called navigation mesh (navmesh), in the editor[1]. This tool would have been really useful but it could not be used. To create the navigation mesh, the terrain has to be a static object, but every object is dynamic in realsite's web player. Therefore, the path finding algorithm has to be developed from scratch using methods available in unity's physics library[2].

Every object in unity has a Transform[3]. It stores the position, rotation and

[1] `http://unity3d.com/support/documentation/Manual/Navmesh%20and%20Pathfinding.html`

[2] `http://unity3d.com/support/documentation/ScriptReference/Physics.html`

[3] `http://unity3d.com/support/documentation/ScriptReference/Transform.html`

scale of the object and allows manipulating their values. If the object has any children, you can loop through them using the Transform of the parent. This was used in the project because each DTU building is composed of many children objects. The transform provides the name of the object that was used to differentiate between buildings and terrain or to detect doors in buildings.

Each building and the terrain in realsite has a MeshCollider[4] attached. This allows to do collision detection using for example the RayCast method[5]. This method was used for detecting the existence of objects between points in the model. SphereCast method was used for the same purpose. These two methods return a struct, which is called RaycastHit. This struct provides the point of collision and the transform of the collided object in case of collision among others.

3.3 Solution designed

3.3.1 System requirements

As it was described before, the routes cannot be represented directly in the model directly. The character would get stuck all the time, moreover there are positions that are unreachable such as the roof of the buildings. Therefore, a system has to be implemented to generate a new route that could be represented in the model. The requirements required for this system were the following:

1. The new route can be represented in the model and looks feasible.
2. The system has to work not only in the DTU 3D model but also in other 3D models that UCS has developed such as the Carlsberg area one. Thus, it has to be easily scalable.
3. The obtained route should be as close to the real route as possible.
4. The system has to be developed in Unity using C#.
5. All the algorithms have to be developed using libraries available in the free version of Unity and compatibles with realsite.dk.

The solution for this problem proposed in this project is a novel algorithm that follows all the requirements proposed above and fixes all the problems presented

[4] http://unity3d.com/support/documentation/ScriptReference/Collider.html
[5] http://unity3d.com/support/documentation/ScriptReference/Physics.Raycast.html

in Section 3.1. The algorithm is a hybrid solution, with multiple steps. It includes not only a map matching algorithm but also other algorithms that can be considered as path finding algorithms. Although, they are not typical path finding solutions, these normally use an environment map and algorithms such as A*.

The system was divided in five different steps that are executed one after another. Thus, the result of each algorithm is the input data for the next one. Although, it is not the most efficient way of solving this problem. It is the easiest to implement. Moreover, this project does not study the performance of the algorithm. This can be done in future improvements of the system. Each of the five different steps of the algorithm are explained in Section 3.3.3 in detail.

3.3.2 Converting from geographic to DTU coordinates and vice versa

As it was described at the beginning of this chapter, the first problem that needed to be solved was the coordinate conversion from geographic to DTU coordinates. This conversion is explained in this section, although it is performed in the server side unlike the other algorithms, which are executed in client side player.

3.3.2.1 From UTM to DTU coordinates

The only reference that was available about the DTU coordinate system is shown in Appendix C. As it can be seen it shows the transformation between UTM and DTU coordinates. Specifically, the UTM coordinates are in zone 32 and uses Euref 89 datum. The article suggests using the Helmert transformation. The Helmet transformation formula that was used to solve the problem was found in [Bes03] and [Teu88].

Using the Helmet formula and the data that is available in Appendix C the following formulas were obtained:

$$\begin{pmatrix} DTU_x \\ DTU_y \end{pmatrix} = \lambda \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} UTM_x \\ UTM_y \end{pmatrix} + \begin{pmatrix} \Delta x \\ -\Delta y \end{pmatrix}$$

$$\begin{pmatrix} UTM_x \\ UTM_y \end{pmatrix} = \frac{1}{\lambda} \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \frac{\sin(\theta)-\sin(\theta)^3}{\cos(\theta)\sin(\theta)} \end{pmatrix} \begin{pmatrix} DTU_x - \Delta x \\ DTU_y + \Delta y \end{pmatrix}$$

$$\lambda = 0.9998133$$
$$\theta = 14.3472^\circ$$
$$\Delta x = 834864.7242936$$
$$\Delta y = 6172294.56110385$$

These formulas were programmed in java and tested during the development of this master thesis. They were found to provide a very accurate transformation of the coordinates.

3.3.2.2 From geographic to UTM coordinates

Once the transformation between UTM coordinates and DTU coordinates were solved, the next step was to solve the conversion between geographic and UTM coordinates. This conversion involves a huge number of calculations, as can be seen in [oOGP11] and [6]. Therefore, the program of the algorithm it was going to be very time consuming. Since it is a very common transformation there are a lot of free resources on the Internet that solve this problem and they are already tested. Thus, instead of fully program the transformation a transformation already programmed was searched on the Internet and modified for the desired specifications in this problem.

The datum used in the conversion that is available in Appendix C is Euref89 which differs from the one that the GPS uses which is WGS 84. A long search in the Internet ended up with the conclusion that Euref89 and WGS 84 are almost the same and that can be assumed that WGS 84 is used when Euref89 is used. As it could be observed with the conversion designed in this project this assumption is completely correct. Since the error in the conversion is negligible.

Some of the conversions found on the Internet can be found in [7] [8] [9]. After

[6] Converting UTM to Latitude and Longitude (Or Vice Versa) `http://www.uwgb.edu/dutchs/usefuldata/utmformulas.htm`

[7] Coordinate conversions made easy `https://www.ibm.com/developerworks/java/library/j-coordconvert/`

[8] Geographic/UTM Coordinate Converter `http://home.hiwaay.net/~taylorc/toolbox/geography/geoutm.html`

[9] Convert Between Geographic and UTM Coordinates `http://www.uwgb.edu/dutchs/usefuldata/ConvertUTMNoOZ.HTM`

trying some of them, it was found that not all of them provides the same results, for instance the code that can be found in IBM website provided results several meters far from the real location, which is why this code was discarded. Geographic/UTM Coordinate Converter provided the best results; the locations were transformed perfectly. This was the selected code, the author claims in the website that the code can be used without restrictions. But the source code is in JavaScript, thus the code was translated from JavaScript to Java. To do that just some variables names and the method headers were modified.

Although, Copenhagen are is in UTM zone 33U, the coordinate conversion, which can be seen in Appendix C, was projected into zone 32U. This is the zone used in the Danish mapping system. The code was modified to fulfill this requirement.

During the development of this master thesis the conversion from geographic coordinates to DTU coordinates was tested many times. It worked perfectly in all the cases.

3.3.3 Algorithms developed to represent the avatar in the model

The five stages, which compose the developed system, are explained in detail in the following sections. They are executed in the same order that they are exposed. In other words, the first algorithm that is executed is "Adding Building Entrances when the user is inside a building" and the last one is "Avoiding hitting buildings". The different algorithms are executed in the order that makes that one algorithm cannot break what the previous algorithms fixed. But the previous algorithms will break thinks that the following algorithms will fix. For example all the algorithms can add new collisions with buildings to the route. That is why the last algorithm to be executed is " Avoiding hitting buildings". This order has the advantage that each algorithm only has to fix its problem and forget about the other ones because they will be fixed later.

To show examples of the different algorithms in the following section, Google maps images are presented. These images were obtained using the Static maps API[10]. But the algorithms were executed in Unity with the 3D model. Thus, Google maps it is only used for the representation of these results. The representation in the model was more complicated to produce. A video would be necessary to show it properly. In the images, the blue line is the route obtained using the Garmin watch. While, the red and the green one are the ones obtained

[10] `https://developers.google.com/maps/documentation/staticmaps/`

with the iPhone. The green one is the result after execute all the algorithms in Unity with the 3D model.

3.3.3.1 Step 1: adding Building Entrances when the user is inside a building

As it was commented in Section 2.5.6, after applying an accuracy filter of 100m and a Wi-Fi positioning filter of 4 positions it is fairly easy to detect when the user was inside a building. The Wi-Fi positions that passed the Wi-Fi positioning filter which are in groups of 4 or more are the ones that were obtained when the user was inside a building in almost all the cases. Using this property, an algorithm was developed that do the following tasks:

1. It detects the Wi-Fi positioning positions in the route.
2. It removes the Wi-Fi positions that are inside a building.
3. It adds the building entrances of the building that is between the first Wi-Fi position that is inside a building and the previous one. It does the same for the first position that is outside a building between the Wi-Fi positioning ones.
4. It adds a jump in the route between the two positions of the doors.

One practical example of this algorithm working can be seen in Figure 3.4, the blue line represents the route received from the GPS watch while the red one and the green one represent the iPhone route before the algorithm and after respectively. As can be seen, the algorithm has added the doors of the buildings 328 and 308 that are the doors that the user uses in the route. Thus, the algorithm works correctly in this example. Other examples have been tested obtaining the same results.

This is the first executed algorithm for several reasons. The first one is that the Wi-Fi positioning positions cannot be filtered out before executing this algorithm. Otherwise some useful positions could be removed and the building in which the user was could be detected wrongly. Moreover, this is the only algorithm that uses the horizontal accuracy parameter. As it was demonstrated before, the correlation between the horizontal accuracy parameter and the real horizontal accuracy for values smaller than 100 m is really low. That makes this parameter useless for the rest of the algorithms. Therefore, this parameter is used in these algorithms and removed after it.

Figure 3.4: One example where two building doors (marked with a yellow circle) have been added to the route because the user was inside the building

Once a building is detected using the RayCast method, the transform of the part of the building that is hit can be obtained using the RayCastHit struct [11]. Using the parent of this transform, all the different parts of the building can be obtained. If the 3D model of the building has building entrance, the name of the entrance part contains the word entrance. This was the used method to detect where the entrances of a building are.

But the center of the doors is usually a point that is not reachable for the character. Therefore, a new point that the character can reach has to be calculated. Four points that are one meter far of the center of the bounds of the door were calculated. The closest point to the previous position of the route was chosen.

Following the implementation of this algorithm, the user jumps from one building door to the next one. Thus, the new route does not contain any information about where the user was inside the building. This is not a big problem, because the location services are really inaccurate inside the buildings. Which building is the user inside is the most accuracy that you can get. Moreover the inside of the buildings is not designed in the model. Therefore it makes no sense to position the user inside the buildings because it is empty.

[11] `http://unity3d.com/support/documentation/ScriptReference/RaycastHit`

3.3.3.2 Step 2: filtering the routes using Douglas-Peucker algorithm

After all the other algorithms were developed, the fast change of heading problem was the most important problem to solve. This problem was explained in Section 3.1.4. The best way of solving it is designing a filter that removes all these positions that are close one to another. Several studies have been done in curve simplification algorithms examples of these are [DD73], [BKR92], [Ram72]. Between them the Douglas-Peucker Algorithm [DD73] was found to be the one that provides better results for geographic information from GPS devices. Further studies using this algorithm has been developed to improve the performance of it [JH]. The first version of the Douglas-Peucker Algorithm was the filter implemented for this project.

The Douglas-Peucker Algorithm is a recursive algorithm that reduces the number of points needed to represent a curve maintaining the accuracy. Other studies in the map matching field used this algorithm [TG11] as well. The algorithm recursively divides the line if the distance between the line segment with the first and last points as endpoints and a point is bigger than ϵ. In our case, ϵ has a value of 5 m. This value was observed to work well for our system. Figure 3.5 shows an example of the algorithm working. The C# implementation of this algorithm developed for this master thesis can be seen below:

```
public static ArrayList douglasPeucker( ArrayList route) {
        double dmax = 0;
        int index = 0;
        for (int i = 1; i< route.Count; i++) {
                double d = perpedicularDistance( (Vector3)route[i], (Vector3)route[0],
                 (Vector3)route[route.Count - 1]);
                if (d > dmax) {
                        dmax = d;
                        index = i;
                }
        }

        ArrayList result = new ArrayList();
        if (dmax > epsilon) {
                ArrayList recursiveResult1 = douglasPeucker(route.GetRange(0, index +1));
                ArrayList recursiveResult2 = douglasPeucker(route.GetRange(index,
                        route.Count -index));
                result.AddRange(recursiveResult1.GetRange(0, recursiveResult1.Count -1));
                result.AddRange(recursiveResult2);
        } else {
                result.Add(route[0]);
                result.Add(route[route.Count - 1]);
        }
        return result;
}
```

If the route contains doors of buildings that have been added by the previous algorithm, the route is split before the Douglas-Peucker algorithm is executed. Each part of the route is treated as a new route. With this system, the Douglas-Peucker algorithm cannot filter out the positions that the previous algorithm added.

After some testing of this algorithm, it was found to produce really good results in our system. The fast change of heading problem was completely solved with

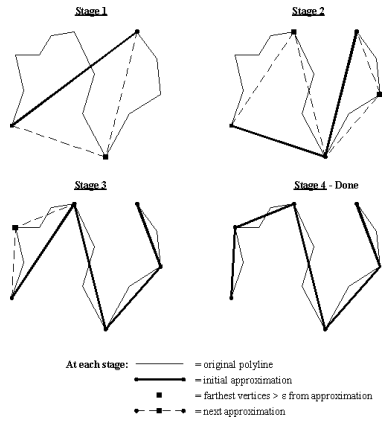


Figure 3.5: Douglas-Peucker algorithm[a]

[a] `http://softsurfer.com/Archive/algorithm_0205/Pic_DP-2.gif`

an ϵ value of 5 m. Moreover, the routes do not lose any accuracy since the main waypoints are not filtered out. The routes are smoother after applying the filter, because the ball change the heading less often. All these advantages produce a route that looks more real. This is exactly what the system was supposed to do.

This algorithm is executed just after adding the building entrance because the sooner the route is filtered the less points have to be handled by the next algorithms. Thus, the next algorithms will be executed faster. Although less efficient the results would have been similar if the douglas filter had been executed just before avoiding hitting buildings algorithm.

3.3.3.3 Step 3: matching the routes with the road network

As it was commented in Section 3.2.2, all map matching algorithms need a road network to work. This road network cannot be obtained directly from the model. Because the terrain is divided in 29 pieces and the same textures that are used in the roads are used in other places in the model. Therefore the road network had to be obtained from other sources. Other option was to generate it manually but UCS wants an algorithm that can be used in other 3D models directly, so this was not an option.

Google maps does not provide an API to access the road networks directly but it

provides the directions API[12]. This API calculates directions between two points using the Google road network. This was studied as a possible solution but it was rejected at the end. The main reason to reject using Google maps API was that the 3D model and the materials used to develop it are way more accurate than the data that Google maps provides. Other solutions like Open street maps or bing maps were studied too but they were found to be less accurate than Google maps in the DTU campus. Moreover, it is always better not to depend on third parties products in your applications. For all this reasons using Google maps or other map service was rejected.

Between the materials used for the development of the DTU 3D model, there is an AutoCAD file that contains detailed information about the different parts of the campus such as buildings, green areas, roads... This file contains segments for the main roads of the campus, the points that compose this segments can be exported into a text file using 3ds Max. This road network was the one used for the map matching algorithm.

Since the road network only includes the main streets in DTU and the user should be able to be placed in all outdoor locations in the model. The map matching algorithms should not match all the positions to the road network. Instead, it should match only the positions that are close enough to one of the segments and have similar heading. This differs from the majority of studies in this field that always try to match all the points of the route. The terminology used to describe the road network is the road network is composed of segments. This segments are composed by two points which are called nodes. Several of this links together compose a street in DTU.

The map matching algorithm developed is an offline algorithm that projects the selected points in the road network. It is a variant of the Algorithm 2 presented by Christopher, David and Alain in [CEW00]. It not only checks how far the projection in the road segment is from the real point but also if the route heading is similar to the road segment one. The different in the route heading has to be smaller than 30 degrees to match the point in the road segment. The distance varies from 20 or 30 m depending on if the previous point has been matched. The projection is calculated as the intersection between the perpendicular segment to the road segment that passes through the point and the road segment. The algorithm works in the following way for each point of the route:

1. If the previous point in the route has been matched. The current point is projected in the same road segment if the distance is smaller than 30m. If the distance is more than 30 m, the point 2 is executed.

[12] `https://developers.google.com/maps/documentation/directions/`

2. First of all, a list of the closest road segments to the current point is generated. This list includes all the road segments which nodes are closer than 300 m to the current point. The point is projected to each of this segments and if the closest of this projection is closer than 20 m. The point is changed for the projection in the road segment.

As it was commented in [CEW00], the calculation of the distance between a point and a segment is different than the calculation of the distance between a point and a line. The problem can be seen in Figure 3.6. This problem was solved calculating the cosines of the angles between the road segment and the segments between each segment node and the point. If one of both cosines or both are smaller than zero that means that it is the first case that can be seen in Figure 3.6.

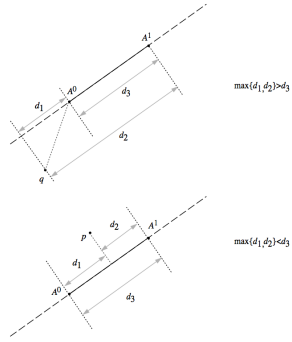


Figure 3.6: The distance between a point and a segment

As can be seen in Figure 3.7, our algorithm matches the route to Nils Koppels Alle street perfectly (green line). This makes the route more accurate because the user was walking in this street when the route was obtained. Moreover, the route looks more realistic in the model because the ball is following a straight line now instead of changing the direction all the time.

3.3.3.4 Step 4: fixing altitude differences

As it was commented before the DTU campus is not completely flat. There are many places that are in different altitudes. There are many places in the campus where there are walls that the character cannot climb. To pass these walls, the user can use some stairs or some buildings that have stairs inside. Our algorithm considers both situations. Therefore, if a change of altitude is

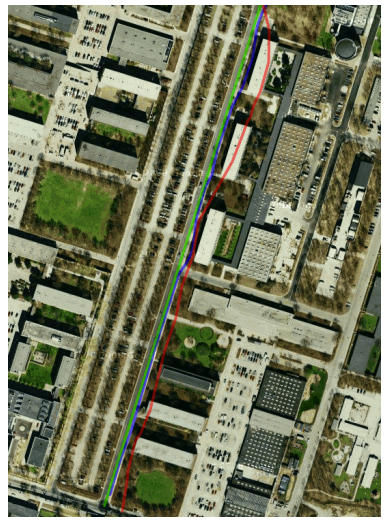


Figure 3.7: Example of route improved with the map matching algorithm

detected between two positions our algorithm try to fix it adding the positions of a stairs or the doors of a building that allows changing the altitude.

As for the stairs, the idea is that the algorithm adds a group of positions to the route to make the character using the stairs. These positions are two or four depending on the types of stairs. The stairs and the walls are included in the terrain object in the model. Thus, there is not way to detect it with Unity. Since there are just seven stairs in the DTU campus, a text file with the points that have to be added for each of these stairs was created. This is not the most scalable way of doing it, but it was the only way of solving this problem. The algorithm uses these stars list to find the closest stairs to the actual point. This list of points that is returned from the text file sometimes has to be reversed to match the altitude of the actual point with the first point of the stairs. An example of this algorithm adding four points to the route can be seen on the left in Figure 3.8 (green line inside of the yellow circle).

Regarding the doors of the buildings, it is worth noting that just few buildings in DTU have doors in two different altitudes, one example of these is the building 341. The algorithm detects the buildings that are around the altitude change using the OverlapSphere method[13]. These buildings are discarded if all their doors are at the same altitude, which is the most common in DTU. One example

[13] `http://unity3d.com/support/documentation/ScriptReference/Physics.OverlapSphere`

Figure 3.8: Example of fixing altitude different algorithm adding the stairs (left) or doors (right)

of this can be seen on the right in Figure 3.8.

Once the closest door and the closest building with doors in different altitudes to the altitude difference point are detected. Their distance to the last point before the altitude difference is compared between the two possible solutions. The closest one is selected if its distance to this point is smaller than 1.5 times the distance between the altitude difference point and the first point to add. If so, these points are added to the route. Otherwise, the algorithm cannot fix the altitude difference and it adds a jump between the two points. With the value of 1.5 times the distance between the altitude difference point and the first point to add, all the altitude differences for all the analyzed routes were fixed correctly. Moreover no false positive was detected.

3.3.3.5 Step 5: avoiding hitting buildings

This algorithm is the last algorithm executed but it is one of the most important ones. The objective of this algorithm is to produce a route visualization free of obstacles. Therefore, a character should be able to follow the route in the model without hitting any obstacle. It is the last algorithm executed because all the points added to the route has to be checked because it may hit some object in the model. Moreover, this algorithm cannot ruin any of the problems fixed by the others algorithms.

The algorithm does two different tasks; on one hand it adds points close to the corners of the buildings to surround them. On the other hand, it removes all the points that are on the top of the buildings. But these points are not removed directly; they are used before being removed. All the objects in the game area are considered as obstacle but the terrain. The problems with the terrain are solved in the previous steps of the algorithm.

The points that are added to the route are 2 m in x and y axes far form the building corners. This solution was chosen because it is the solution that requires fewer resources to be obtained. Other solutions such as the closest path to this corner would have been better. But the same problem than for the roads and the stairs arises, as the model is designed it is impossible to detect this in Unity. Another external source has to be used to get this information.

Unlike some of the other algorithms, this algorithm works only with information obtained from the 3D model in Unity using the Physics class. The main methods used are Raycast and SphereCast[14]. Both check if there is any obstacle starting in a point and following a direction; if so some information of this obstacle is returned.

The algorithm should be able to add any number of points to surround a building. It is very common to find several buildings that are connected in DTU. Thus the algorithm has to be able to solve complex buildings shapes and not only rectangular buildings. Furthermore, sometimes the positions can be far one for another, the algorithm should find a path surrounding the buildings in these cases as well. For all this reasons, a recursive algorithm was designed to solve this problem.

This program is called for each point in the route. The point (-1, -1, -1) is a jump between two points. As can be seen in the code below, if the current or the previous points are a jump the point is just added to the route. After that, it checks if there is an obstacle in the line between the current point and the previous one. If so, it tries to find a solution. It tries to add the corners of the obstacle that was hit. If some of these corners are free of obstacles from the point previous to the hit the one that produces the shortest path is added. If all the corners add an obstacle to the route, the corners of these obstacles are tested. For each point that is added to the route, the same algorithm is executed again recursively.

The code of the recursive algorithm programed can be seen below:

```
void addPointToRoute (Vector3 point, ref ArrayList route)
{
```

[14] http://unity3d.com/support/documentation/ScriptReference/Physics.SphereCast.html

```
        if (point == new Vector3 (-1F, -1F, -1F)) {
                route.Add (point);
                return;
        }
        if (route.Count > 0) {
                if ((Vector3)route [route.Count - 1] == new Vector3 (-1F, -1F, -1F)) {
                        if (!pointInsideBuilding (point)) {
                                route.Add (point);
                        }
                        return;
                }
        }

        RaycastHit hit, hitn, nhit;
        bool obstacle = obstacleBetweenPoints ((Vector3)route [route.Count - 1], point,
         out hitn);
        if (obstacle && nameOfBuilding (hitn.transform.name)) {
                hit = hitn;
                Vector3 newPoint = new Vector3(-1F, -1F, -1F);
                int i = 0;
                while (newPoint == new Vector3( -1F, -1F, -1F)) {
                        if (route.Count > 1) {
                                newPoint = pointToNotHitBuilding
                                ((Vector3)route [route.Count - 2],
                                                        (Vector3)route [route.Count - 1],
                                                        point, hit, out nhit);
                        } else {
                                newPoint = pointToNotHitBuilding
                                (new Vector3 (-1F, -1F, -1F),
                                                        (Vector3)route [route.Count - 1],
                                                        point, hit, out nhit);
                        }
                        hit = nhit;
                        if (i == 5) {
                                break;
                        }
                        i++;
                }
                if (!((Vector3)route [route.Count - 1] == new Vector3 (-1F, -1F, -1F) &&
                        (newPoint == new Vector3 (-1F, -1F, -1F)))) {
                        route.Add (newPoint);
                }
                if ((newPoint == new Vector3 (-1F, -1F, -1F) &&
                        !pointInsideBuilding (point, out hitn))) {
                        //if a new point can not be found the route will jump
                        route.Add (new Vector3 ( -1F, -1F, -1F));
                        route.Add (point);
                        return;
                }
                if (!pointInsideBuilding (point, out hitn)) {
                        //remove the points that are inside a building
                        addPointToRoute (point, ref route);
                }
        } else {
                if (!pointInsideBuilding (point, out hitn)) {
                        route.Add (point);
                }
        }
}
```

The pointToNotHitBuilding method returns the closest corner to the previous and the next point of the object that was hit that is free of obstacles with the previous point. If this corner does not exist, it returns (-1, -1, -1) and the returned hit is the obstacle that was hit while testing the corners.

The Figure 3.9 shows two examples of the execution of this algorithm. The colors used for the lines is the same than in the previous sections. As can be seen the algorithm generates a free of obstacles path, although the positions are far one to another and on the top of buildings.

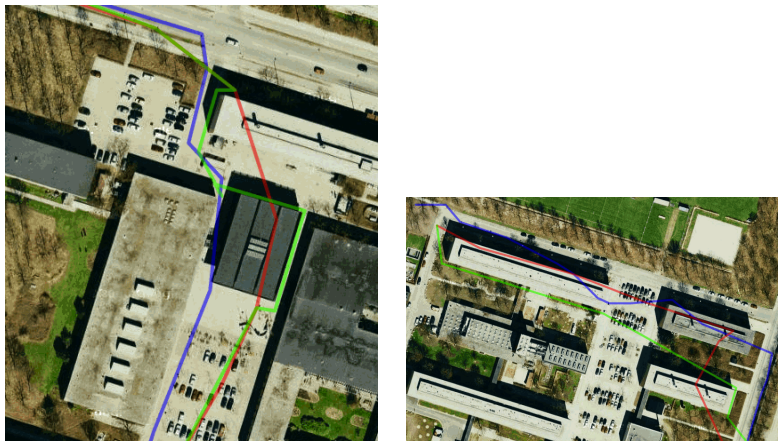


Figure 3.9: Examples of execution of the algorithm avoiding hitting buildings

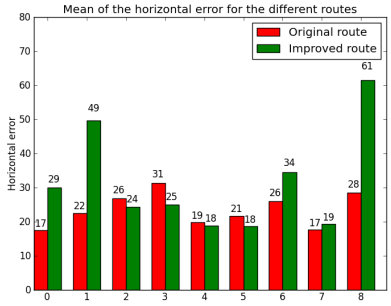
3.4 Results

The algorithm satisfies the requirements stated in Section 3.3.1 for all the analyzed routes. It produces a feasible route that can be represented in the model without getting stuck. Although, the main requirements are fulfilled, the new routes should be more accurate than the original ones. This is going to be analyzed in this section.

In order to quantify the inaccuracy of the improved routes a new error has to be calculated. Three solutions were analyzed to solve this problem, using Hausdorff Distance [FDA08], Fréchet distance [AM06] or the same solution that was done in Section 2.5.3. The last option was the selected one. This solution measures not only the horizontal error but also the time error of the positions. Moreover, the results of these error calculations can be directly compared with the ones obtained in Section 2.5.3. It is worth noting that this algorithm only tries to improve the horizontal error and not the time error.

But this solution has a problem; after the algorithm is executed the route has points that were not present in the original route. Therefore, these points do not have the time that the position was received. This time has to be calculated before the error calculations. Two solutions were evaluated to solve this problem. The first one was to measure the average speed of the whole original route. Assuming that the speed is constant during the whole route, a new time stamp is calculated for each position in the improved route. This solution produces really

high errors. Although the speed of the user is approximately constant during the route, the speed in the received route is not constant as was previously demonstrated. This is the reason why the calculated error was high for this solution. The solution that was adopted was to calculate a timestamp just for the positions that are added by the algorithm. These timestamps were calculated assuming constant speed between the added positions and using the time that takes in the original route to do this part of the route.



Route name	Number
from 308 to 342	0
from 341 to 101	1
from 343 to 308	2
from library to 229	3
from 115 to library	4
from 229 to 115	5
from Nordvej to 302 biking	6
from Knuth-Wintherfeldts to Nils Koppels Alle walking	7
from Nils Koppels Alle to Knuth-Wintherfeldts biking	8

Figure 3.10: Comparison of horizontal error between original and improved routes

The error calculated for each route after and before the algorithm is applied can be seen in Figure 3.10. The detailed results of the algorithm for each route can be seen in Appendix D. The routes are filtered with an accuracy filter of 100m and a Wi-Fi filter of 4 samples, which was found to produce the best results. As can be seen, the error is improved in 4 of the 9 routes. At the first glance these results seem not to be very promising. But as it was commented before, this algorithm tries to improve the horizontal error and not the time error.

In the routes which are called “from Knuth-Wintherfeldts to Nils Koppels Alle walking” and “from Nils Koppels Alle to KnuthWintherfeldts biking” the mean of the error increases 1.55m and 33.02m respectively. The routes that the algorithm returns in these two examples are even more accurate than the ones that the Garmin watch obtained since the routes are matched with the road, which the user was during all the time. Therefore, the increase of the error in these two routes is caused because the position time stamps are not the correct ones. This problem cannot be easily solved and it is not important for the application of our system. It does not really matter that the timestamp are incorrect since they are not used for the representation of the users in the model.

The routes “from 308 to 342” and “from 341 to 101” increases its average error

12.4m and 17.2m respectively. As can be seen in Appendix D, the routes that the algorithm returns unlike the original ones can be represented in the model. However, some parts of them differ from the original ones. This last fact and the time error produce this increase in the mean error. Other route which error increases after the algorithm is executed is "from Nordvej to 302 biking". Analyzing the returned route can be seen that the route is closer to the real route than the original one. Therefore, the timestamps were correct in this case as well.

As it has been explained, only two routes are further form the original one after executing the algorithm which are "from 308 to 342" and "from 341 to 101". During these routes, the user went inside buildings several times and the results that the iPhone provided are really inaccurate. The other 4 routes that have not been examined in this section are closer to the real route than the original ones, as can be see in Appendix D. Some parts of them are even more accurate than the ones that the Garmin watch obtained. Therefore, it can be concluded that the algorithm improves the horizontal error in 7 out of 9 routes. The time error is worsening the horizontal error in all the cases. Moreover it provides a route that can be represented in the model in all the cases. These are very promising results, hence the intention of Utopian City_scape is continuing developing this algorithm in the future.

3.5 Future improvements

Although the algorithm fulfills the requirements for all the analyzed routes, there is plenty of room left for future improvements which Utopian City_scape would like to develop in the future. The improvements can be divided into two groups, improvements in the data that the algorithm uses and improvements in the algorithm itself. The improvements in the data that the algorithm uses are:

- As the terrain is designed now in the model, the walls are included in it. This makes impossible to differentiate between the ground and the walls when the object is hit. To improve this, the walls should be different object than the terrain.
- There are buildings in the model which doors are not separated for the other parts of the building. In this situation, it is impossible to detect where the entrance of the building are. This should be fixed.
- The road network that was used only included the main roads in the campus. Increase the number of roads in this network will increase the accuracy of the algorithm.

Next, the improvements that can be developed in future versions of the algorithm are listed:

- The algorithm was not designed taking into account performance. Therefore, the performance of the algorithm can be studied and improved.
- The algorithm does not check the positions that have been added to the route. The positions that should not have been added should be detected and removed. An example of this problem can be seen in the route "From 308 to 342".
- The algorithm avoiding hitting buildings does not evaluate different groups of positions to add. It just evaluates which position to add at each time between the lists of possible ones. It would be better if this algorithm could compare between different groups of solutions and select the best one.
- When the user is detected to be inside a building the algorithm add two entrances of buildings and a jump between them. These entrances can be from different buildings. The algorithm does not check if the buildings are connected in case the doors are in different buildings. This should be check in order to obtain more feasible results.
- The map matching algorithm does not detect if the roads are connected between them. This should be detected and taken into consideration.
- Several parameters have to be introduced in the algorithm manually. These parameters have been set with reasonable values and are found to work well. But the accuracy of the algorithm would be better if the possible values for these parameters will be analyzed in more detail.
- Nine routes have been used to test the algorithm. More routes have to be used to find possible bugs and improvements.
- The algorithm only makes the route crosses buildings to avoid an altitude difference or when Wi-Fi positioning is detected. The algorithm should be able to detect when crossing a building is better option that surround it.
- In the route "from Nordvej to 302" the algorithm "Fixing altitude differences" produces a false positive. Changing the parameter that controls the maximum supported altitude difference and slope between two points causes other altitude differences not to be solved. This problem has to be solved implementing a more sophisticated detection algorithm.

Chapter 4

Final functional system

This section describes the final system implementation. How the different parts that were explained in the previous sections were connected to produce a fully functional system. The system has been tested under several scenarios. It has been found to follow all the required specifications. Although, it is worth noting that this system is more a prototype than a final deployable system. This project pretends to be a proof of concept which will allow UCS to develop more advanced services in the future using this technology.

4.1 System architecture

As aforementioned in this report, the objective of this master thesis is to develop a tracking system using an iOS device in the UCS platform realsite.dk. The developed system is composed of three main parts, the mobile application, the server side with the database and the website. The mobile application was fully developed as part of this master thesis. Moreover, parts of the system in the other two parts were developed as part of this master thesis as well. The functions to convert from geographic coordinates to the reference system of each model which is explained in section 3.3.2 were developed for the server side. As for the website, the algorithms explained in section 3 were developed. People

in UCS developed the rest of the system because it was highly connected with their systems and they are the experts on that.

As can be see in Figure 4.1, the mobile app and the website running in the user PC cannot communicate directly. Therefore the server is necessary to allow this communication and to store the routes in a database. The communication of both the mobile application and the website with the server are made mainly using HTTP GET and POST requests. The system is developed to work over the Internet. Therefore an Internet connection is necessary in both the mobile and the browser side. The system is made to work in real time. Thus the positions have to be sent to the server as soon as they are available in the phone. In the next sections the three parts that the system is composed of are going to be analyzed in more detail.

Figure 4.1: System architecture diagram[a]

[a] http://cdn1.iconfinder.com/data/icons/database/PNG/512/Database_1.png http://www.jailbreakyourphone.net/wp-content/uploads/2011/07/iphone.png http://aux.iconpedia.net/uploads/1360992576.png http://upload.wikimedia.org/wikipedia/commons/thumb/c/c1/Computer-aj_aj_ashton_01.svg/2000px-Computer-aj_aj_ashton_01.svg.png

4.2 Mobile application

The app that was developed for the location services accuracy analysis, presented in section 2.3.2, was modified to develop the final app. The user interface was changed and additional functionalities were added to the app. Moreover, the communication with the app and the server was improved to make the system less app dependent. One of the requirements of the project is that it has to be easily scalable. Therefore, the mobile application fulfills this requirement no change are needed to support future models added to realsite.dk. The application was developed for iOS 5.1 and tested in an iPhone 3GS.

4.2.1 Function

The main function of the mobile application is to obtain, filter, store and send the user's locations. The mobile phone is the only device that is location-aware in all the system. The tracking is not always enabled; the user decides whether the tracking is enabled or disabled pressing one button in the main windows of the application. Each route has associated a route description and a picture, which have to be defined by the user before starting the tracking. The picture has to be taken using the camera in the phone. It is stored in the internal memory of the phone. Thus, the same photo is used for every route until the user takes a new one.

The application detects which site the user is. If the user is not in a place where there is a 3D model available, the tracking cannot be started. This functionality is necessary because one of the parameters that are stored for each route in the server is the site where the route was tracked. After this site recognition, the same tasks in the application are performed in every place.

All the positions that are obtained and sent to the server are stored in a database in the phone. Therefore, the application allows the user to explore the previous routes. The routes are shown in a list ordered by date. A map of the route is displayed in the screen if the user selects a route. Other window shows additional information of the route such as the number of used positions and the duration.

4.2.2 Obtaining and processing location data

As it was explained in section 2.2, the location of the user in iOS only can be obtained using the Core Location framework. During the chapter 2, *CLLoca-*

tionManager parameters such as the *desiredAccuracy* and *distanceFilter* were studied. Moreover, several filter solutions were analyzed to remove the not accurate enough locations. The conclusions of this chapter were used in the development of the final app.

Following the results of these sections, the final app has *CLLocationManager desiredAccuracy* set to *kCLLocationAccuracyBest*. Furthermore, the *locationFilter* is set to 1 meter to receive just the positions that are new. Not all the locations that the Location Manager provides are used. The locations are filtered using the filter that was presented in section 2.5.4. As it was demonstrated in this section, the filter that provides better performance was the filter with error filter of 100 meters and Wi-Fi filter of 4 locations. The error filter removes the routes which predicted horizontal accuracy is bigger than 100 meters. While the Wi-Fi filter filters out the Wi-FI positioning locations that are in groups of less than 4 positions after the error filter.

The same positions that are sent to the server are stored in the internal memory of the device in a database. The database is managed by the app using Core Data as described in Chapter 2.3.2. The database has two tables one table for the routes and other one for the positions. Each route in the route table has associated several positions in the positions table.

Each time the application is opened the list of available sites is downloaded. This list includes the coordinates of the corners that include the area. Using these coordinates and the first accurate coordinate available the site where the user is placed is detected.

4.2.3 User interface

The user interface of the app is composed of a tab bar, which allows the user to switch between the two main windows, Start and Routes. In the start windows, the user can switch the tracking on and off and changes his avatar photo. While in the routes one, all the routes that have been taken with the app are presented in a list. Pressing on it leads into a map of the route and info of it.

As can be seen in Figure 4.2, the start tab presents a window which is compose of the elements which are needed to start a new tracking session. The user can find a text field to write the route title. If no title is specified, the route title is set to "No title specified". The next button starts or stops the tracking. If the button is green as can be seen in the picture in the middle in Figure 4.2, the tracking is started when the button is pressed. While if the button is red as can be seen in the right picture, the tracking stops. The change of color is

accompanied with a change of the text that is written on it.

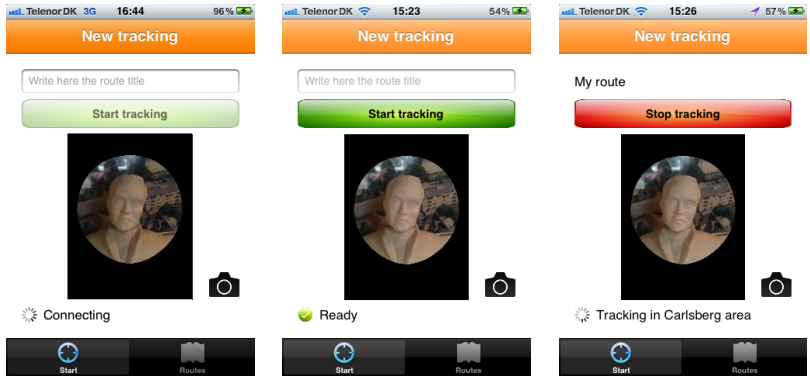


Figure 4.2: Different states of the start window

Below the button, the avatar picture can be found. The user can take a new avatar picture clicking the camera icon which is on the right of the photo. The screen while the user is taking the photo shows an oval hole that has the shape of the avatar's head as can be seen after the photo is taken in the start window.

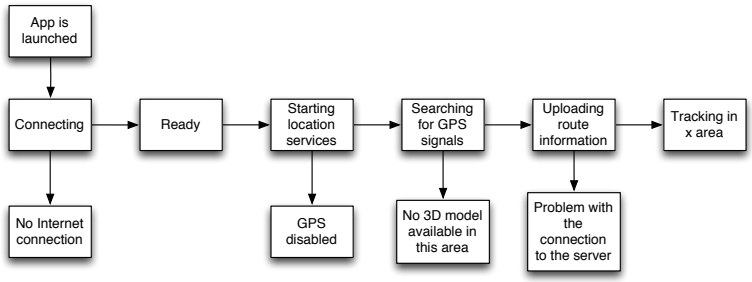


Figure 4.3: States diagram of the application

Under the avatar picture, there is a line of text with an icon on the left that shows the status of the app. As can be see in Figure 4.3, the app has ten states that have associated one of the four different icons. These four different icons indicate either ready or loading or warning or error. In Figure 4.2, the loading and the ready one are shown. The list of the ten different states and what they mean is listed below:

- Connecting: this is the state that is shown when the app is started. During this state the app downloads the list of the different sites where there is a 3D model available. If this download fails, the app assumes that there is not Internet connection available and switches to this state. If the download works, the app switches to the Ready state.

- No Internet connection: this state indicates that the list of the sites could not be downloaded. The tracking cannot be started in this state because the application has to tell to the server in which site the tracking is done. Without the list of the sites this parameter cannot be calculated.

- Ready: it means that the list of sites has been downloaded successfully and the tracking can be started by the user. This state is the state that the app has when a tracking session finishes.

- Starting location services: this is the first state that the app has after the tracking session is started. During this state the database is opened and the Location Manager started. Opening the database usually takes around 1 or 2 seconds, once the database is opened the Location Manager is started immediately. After this state the app switches to the next state of this list.

- Searching for GPS signals: the application stays in this state until a location that passes the filter is received. This means that a location with a horizontal accuracy better than 100 m has to be received in order to switch to the next state. This location is used to calculate in which site the user is.

- Uploading route information: in this state the route information is uploaded to the server. The route information contains the route title, the avatar picture and the site id. This state takes usually several seconds because the image size is high and it takes time to upload it to the server.

- Tracking in x area: this state indicates that the tracking session has started without problems. The positions are received and sent to the server in real time during this state. If the Internet connection is lost during this state, the positions are stored and sent to the server when the connection is available again.

- GPS disabled: the app enters in this state when the location services are disabled or the user does not allow the application to obtain his location.

- No 3D model available in this area: if the user tries to start a tracking session and he is not in a place where there is 3D model available. The app enters in this state.

- Problem with the connection to the server: this state is caused when uploading of the route information to the server fails.

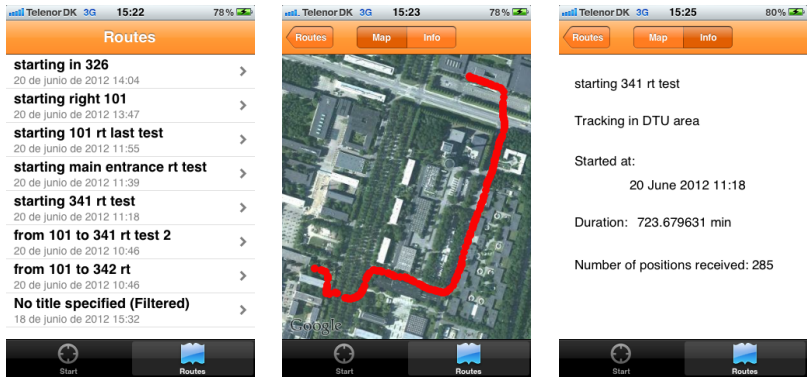


Figure 4.4: All the windows that are shown in the route tap

As can be seen in Figure 4.4, the Routes tap shows a list of the tracking sessions that have been recorded with the app since the app was installed. This list shows exactly the routes that are stored in the device internal memory database. These routes should be the same that are stored in the Internet server database. Although, the app handles connection problems during the tracking, it can happen that some positions are lost during a tracking. This can happen if the app is terminated during a tracking while there is not Internet connection available. This is unlikely but it can happen.

Tapping in one of the route leads to a windows where a map of the route or information about it are shown. As can be seen in Figure 4.4, the two buttons that are in the navigation bar which is in the top of the screen allows the user to switch between the two windows. The map window, which can be seen in the center, shows a map centered in the route. Each point that has been sent to the server is represented with a red dot. Tapping in these dots shows a callout with the timestamp and horizontal accuracy of this position. The info window shows information about the route such as the title, the site of the tracking, the date when route was started, the duration and the number of positions that have been received.

4.2.4 Communication with the server

Since the app is designed to track users in real time the communication with the server is a crucial part of it. The app communicates with the server using HTTP POST and GET requests. While in the GET request only the URL and the headers are sent to the server, the POST request also includes a message body. The app sends all the parameters in the URL but the picture that is sent in the body of a POST request.

IOS provides several classes to handle HTTP requests. *NSURLConnection* class has been used for uploading the photo. For the other requests, the method *stringWithContentsOfURL* of the class *NSString* has been used since all the parameters were sent in the URL and the server just returns a string. The class *NSURLConnection* with *NSMutableURLRequest* allows the developer to set between others the type, the header and the body of the HTTP request. If the body of the HTTP request has to be sent, these methods should be used to do so. The other method used in the app hides the HTTP protocol for the developer and makes an HTTP GET request. This method can be where all the information is sent in parameters in the URL.

The communication with the server is done asynchronously in the app. This means that all the HTTP requests are not done in the main thread. This is basic in a mobile application since the main thread is in charge of the user interface. If the HTTP requests are done in the main thread, the user interface will be stuck until the request receives a response. This leads into a really poor user experience. That is why the HTTP requests are executed in a secondary thread using either asynchronous request with a completion handler or synchronous request that are executed in other queue using the dispatch_async framework.

The application makes three different types of requests to the server which are listed below in the same order that are executed:

- Download the list of sites: this is the first request that the application executes when it is launched. The server responses to this request a list of the available sites. This list contains for each site its name, site ID and the coordinates of the corners of the polygon that contains the site.
- Get the route ID: this request is sent when a tracking session is started and is a HTTP POST. This request sends to the server the route title, the site ID and the route picture. While the route title and the site ID are sent as parameters in the URL, the picture is sent in the body. The sever returns the route ID for this session. This route ID is used for each location that is sent to the server to identify the route that it belongs to.

- Upload the locations: this request is sent each time the application receives a new location that passes the filter. All the parameters are sent in the URL. The parameters that this request sends are longitude, latitude, altitude, time interval, horizontal accuracy and route ID.

As it was commented before the photo is uploaded at the same time that the route ID is obtained using a HTTP POST request. The raw JPEG data of the photo is placed in the body of the request. But this body has to have a special format to be understandable by the web application. This format is the format that the HTML forms have.

4.3 Server side

The server side executes the web application that handles all the requests that have been explained in the previous sections. Moreover it stores the routes and the positions in a database. The player in realsite.dk has to access the data that the application stores in the database. Therefore the server side has to provide an interface for this communication as well which is explained in the next section.

The routes are stored in a MySQL database. The database has two tables one for the routes and other one for the locations. Each location in the locations database belongs to one route from the routes table. The locations are stored using the class point of the Spatial Extension of MySQL[1].

The positions are stored in the database in geographic coordinates. But as it was explained before in this report the models have their own coordinates system that is always a transformation of the UTM coordinates of the place. Therefore, the coordinates transformation that was explained in Chapter 3.3.2 is performed by the server when the player requires the coordinates of a route.

4.4 Player in realsite.dk

The player in realsite.dk is the interface that shows the routes that have been obtained with the mobile application in the 3D model. This player was developed using Unity. It is accessible using every browser with unity plugin installed,

[1] `http://dev.mysql.com/doc/refman/5.5/en/gis-class-point.html`

therefore is executed locally in the computer of the user. The player used to be private; a realsite account was necessary to see it. But since April 2012 everyone can see the player with the DTU 3D model in www.realsite.dk/DTU/public. The public and the private versions of the player are the same. Thus, the functionalities that are explained in this section are available in both versions.

The routes are not shown in the model when it is started. Instead, the key L has to be pressed to activate this functionality. After the letter L is pressed a pink ball is created for each route that is stored in the database. As it was explained before, the balls cannot be moved in the model using the points received from the phone directly. Therefore, the transformation that was designed as part of this master thesis and presented in Chapter 3 is executed for every route. The balls are moved following the route that the algorithm returns. The algorithm worked perfectly for new routes that were obtained using the whole system.

As can be seen in Figure 4.5, users are represented as a ball since it would be very resources consuming using a normal character for each route. Thus, a simple object has to be used to represent the users. The ball was selected because is one of the default game objects in Unity. But in future developments of the system the ball is going to be changed for a 2D not animated character. This character will have a body shape and contain the pictures that the mobile application sent in his face. The color pink was selected for the balls because is one of the easiest colors to differentiate inside the model.

Figure 4.5: Two screenshots of realsite with the system activated

The camera in the player is always attached to the character. Therefore the character has to be moved around the model manually to see the different routes, this can be seen in Figure 4.5. There is no way to search for the different balls but manually. As the system is implemented the balls are placed in the model but there is no way to see information about the different routes or to find them. Future versions of the system should improve this.

The balls never stop; they are continually moving form the beginning of the route to the end. Once the end is reached the ball starts in the beginning of the

route again. This implementation was chosen mainly for testing purposes. The system was tested in real time; the new points are loaded in the player as soon as they are stored in the database in the server. The algorithm to improve the routes is just executed for the new positions and not for the old ones. Everything worked fine in the real time testing; the delay of the whole system is around 1 minute, which is pretty promising for future development of the system.

As it was commented before, a future version of the characters that represent the routes in the model are supposed to include the picture that is taken with the mobile application in their heads. For testing that the communication of the pictures is working, the actual version loads the routes picture in the balls. As can be seen in the Figure 4.5, the communication is working perfectly.

4.5 Potential improvements

As it was commented before in this report, this system is a prototype, which can be used to develop advanced tracking services. It is a proof of concept using Utopian City_Scape products. Thus, there are many aspects that can be developed in the future. The improvements can be either in the mobile application or in the player. The mobile application should be improved mainly in the following points:

- Login: the application does not have any authentication. This is a big issue because every user that downloads it can upload infinite number of routes to Utopian City_Scape server. That is why the application has not been deployed in the app store. But this login can be implemented in the same way that the realsite login and it should be really easy to deploy.
- Design: since the application is a prototype the design is not an important part of it. It could be improved with some graphics and redesigning the layout of the different windows.

Regarding the representation of the routes in the player in realsite.dk, the following improvements could be done:

- Improve the algorithm for improving routes: although the algorithm returns a route free of building obstacles in almost all the cases. The conditions that detect the several problems can be improved to be more accurate. Moreover it sometimes adds some corners of buildings that should not be added.

- Divide the different parts of the terrain into different object: as it was commented before in this master thesis the walls and the outdoor stairs are integrated in the terrain object. This makes the detection of walls very complicated and inefficient. As it is now, the algorithm does not detect walls, it detects altitude difference. If the walls were different objects, the detection of the walls would be very easy and avoid them could be done in the same way that was done with the buildings.
- Design an user interface to interact with the different routes: as it is implemented now, the only user interaction with the routes is done by pressing key L. A new user interface should be designed where the user can select which routes should be visualized. Moreover, some way to find the routes in the model should be available as well. As it is now the user has to move the player around until he finds a ball.
- Improve the road network: the road network that was used for the map matching was generated automatically. Just the main streets of the campus are available in this network. A more accurate network will improve the results of the algorithm.

The final system has been tested several times. But it is not enough to assume that it has no errors. More tests should be performed in order to find bugs and fix them.

Chapter 5

Conclusion

Firstly, this master thesis studies the accuracy of the iOS location services. The study shows that the statics positions are up to 8 m accurate in average. Moreover some configuration parameters were found to have no influence over this accuracy. Other additional parameters that are provided apart form the geographical coordinates such as the time stamp and the predicted horizontal accuracy were analyzed, they were found to be more inaccurate than the coordinates. As the study demonstrated, the accuracy is not the best quality of the location services in smartphones but they are really fast and available almost everywhere. These two characteristics are very important in a mobile device, even more important than the accuracy. The study shows that the location services are able to provide an accurate position within 15 seconds in almost all situation and 10 seconds in the majority of them. The availability of iOS positioning in the campus area was found to be almost 100%.

Not only static positions were analyzed but also positions in motion. The error in these cases was calculated using as reference the positions that a more accurate device provided. The mean error in this case is bigger than in the case of the static positions because this technique introduces some error. The mean error for all the analyzed routes was 27.13 m. This error was reduced to 23.89 m after filtering out the inaccurate positions. Inaccurate positions are the ones that have a predicted horizontal accuracy bigger than 100 m. The WiFi positioning positions that appear in groups of less than 4 positions are considered inaccurate

as well. After applying this two filters detecting when the user enters a building is as easy as detect the remaining positions obtained using WiFi positioning.

After this study, a novel system to represent inaccurate routes obtained with an iPhone in a very accurate 3D model was developed. As it was demonstrated, the characters cannot be moved directly in the model using the row GPS data. A transformation of this data has to be performed. This transformation is done by a system, which is composed of five different stages. These are executed one after another. Each of these stages improves one problem of the route without ruining the work of the previous ones. Although the system cannot be considered as a path finding algorithm, the problem that solves is related to a path finding problem. One of the algorithms is a map matcher. Moreover the algorithm was developed in Unity using the Utopian City_Scape 3D model of DTU.

The system improved the average of the horizontal error in four of the nine analysed routes. After a deeper analysis of the result routes seven of the nine routes were found to be closer to the real ones after applying the algorithms. Moreover for all the cases the system returns a route that does not get stuck. Although these results are very promising, this system pretends to be a proof of concept. Utopian City_Scape has already stated the intention of developing further this system in the future.

The final functional system is composed of the iOS app, the webserver with the database and the 3D model player. The mobile application obtains the GPS positions. These positions are filtered and stored in the device internal memory before they are sent to the server. The server stores this positions in a database and convert them into the 3D model coordinate system.

Appendix A

Product Brief iPhone A-GPS chip

[1]

[1] http://www.infineon.com/dgdl/PMB2525-Hammerhead+II-pb.pdf?folderId=db3a304316f66ee80117824fc0d71e07&fileId=db3a304316f66ee8011782518d4a1e08

Hammerhead™ II

PMB 2525

The Industry's Highest Performance GPS Receiver in the smallest package yet.

Hammerhead II delivers all the performance of its predecessor, while achieving a new benchmark for high performance Positioning & Navigation integration into mobile devices.

USING CHIP scale packaging (CSP) technology, its size (3.59 x 3.75 mm²) now offers possibilities of integration into the miniaturizing world of mobile handhelds.

Hammerhead II integrates a high performance A-GPS baseband processor and a low-noise GPS RF front end. It comes packed with new software features such as advanced multi-path mitigation that avoid large errors in urban environments caused by reflected signal-in buildings and other structures.

Key Features and Benefits

- Single-chip minimizing board space (< 50 mm² PCB area for complete A-GPS solution)
- Advanced low-power 0.13 µ RFCMOS technology with smart power management
- Real-time hardware correlator engine (fast acquisition and high sensitivity)
- High sensitivity, -160 dBm, enabling indoor and deep urban operation
- Built-in voltage regulators supporting single-power supply source
- Multiple-mode operation
  - MS-based (calculation of position in mobile handset)
  - MS-assisted (calculation of position in base station)
  - Autonomous (no assistance by network)
  - Enhanced autonomous (using four day assistance data)
- Multiple protocol operation
  - Control plane (RRLP & RRC)
  - User plane (SUPL)
- A-GPS control software enables non disrupting call flow and ease integration
- Standard compliant (exceeds requirements for 2.5 G and 3 G networks)

Note: The Hammerhead trade mark is owned by Global Locate, Inc

Specification

- -160 dBm sensitivity
- Time-to-First Fix (Mobile-Based)
  - 1 second@5 m accuracy (cold start, -130 dBm)
- 2 m steady state accuracy
- Reference frequencies: 10 – 40 MHz
- Assistance data standards support
  - UMTS/GSM: 3GPP TS 25.331, TS 44.031, and OMA SUPL
  - CDMA: 3GPP2 C.S0022-0-1

Devices

- Mobile phones
- Smartphones
- PDAs
- PND (Personal Navigation Devices)

Applications

- Emergency Assistance (E911, E112)
- Navigation: Point-to-Point, POI, Business finder, Real-time Traffic information
- Child safety and Friend-finder
- Fleet and workforce management
- Location Games

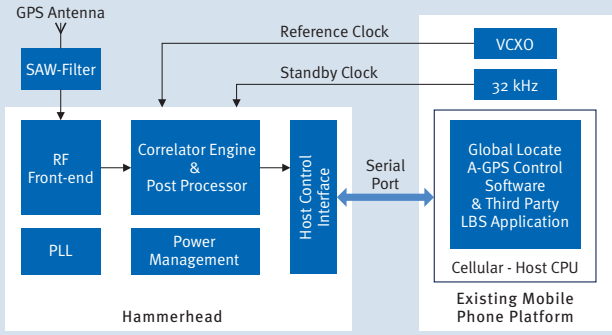
www.infineon.com/gps
www.globallocate.com

Communication Solutions

Never stop thinking

Product Brief

Block Diagram



- Cost-optimized architecture for integration into mobile phones
- No additional external CPU or memory required
- Both A-GPS control software and LBS (location based services) application executed on host CPU of cellular processor
- Serial host interface (UART, SPI or I^2C)
- No additional TCXO required, uses host 32 kHz standby clock & reference clock (10 to 40 MHz)

Product Summary

Product	Sales Code	Package
Hammerhead II	PMB 2525	SG-UFWLB-49 (3.59 x 3.75 mm^2) lead-free/halogen-free

Application Examples

- Personal Navigation: Never get lost again in foreign cities. Your mobile phone is leading the way to your destination, whether you start indoors or outdoors. Avoid being delayed in traffic jams by utilizing real-time traffic information.
- Location-based Services: Finding the way to points of interest simplifies your life. Take the short way to restaurants in your neighborhood, the nearest gas station or metro station. Use your LBS application indoors to start planning your trip before you get in your car.
- Emergency Assistance: A-GPS-enabled mobile telephones are in demand to allow accurate tracking of emergency calls placed from mobile telephones by the end of 2005. Hammerhead II will enable mobile users calling emergency telephone codes to provide emergency services with very accurate location information, both from indoors and outdoors.
- Friend-Finder and Child-Safety: Meeting with friends and family is now an easy task by sharing your personal location. Keep an eye on the current position of your children and keep them away from dangerous places.

For further information on the Hammerhead product please contact Global Locate (www.globallocate.com) or Infineon Technologies (www.infineon.com/gps).

Advantages

Hammerhead II is based on Global Locate IP and state-of-the-art Infineon RFCMOS IP and process technology. This ground breaking chip is the key for enabling location-based services such as emergency assistance and personal navigation in deep urban canyons, in moving vehicles, and indoors.

Hammerhead II uses the proven host-based architecture pioneered by GL as the best fit for mobile devices as proven in mass market 2.5 G and 3 G devices shipping now. GL host-based architecture leverages some of the resources already existing in the mobile device without imposing a big CPU load (~ 3 – 6 MIPS) or any real time requirements. Hammerhead II uses standard serial communication interfaces with speeds as low as 38,400 bps. Global Locate's Host-based architecture yields the lowest system cost solution as well as the smallest footprint without compromising performance.

How to reach us:
http://www.infineon.com

Published by
Infineon Technologies AG
81726 Munich, Germany

© Infineon Technologies AG 2006.
All Rights Reserved.

Legal Disclaimer

The information given in this Product Brief shall in no event be regarded as a guarantee of conditions or characteristics ("Beschaffenheitsgarantie"). With respect to any examples or hints given herein, any typical values stated herein and/or any information regarding the application of the device, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

Information

For further information on technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies Office (www.infineon.com).

Warnings

Due to technical requirements components may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies Office.

Infineon Technologies Components may only be used in life-support devices or systems with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system, or to affect the safety or effectiveness of that device or system.
Life support devices or systems are intended to be implanted in the human body, or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

Ordering No. B130-H8965-X-X-7600
Printed in Germany
PS 0107.75 nb

Published by Infineon Technologies AG

Bibliography

[AM06] Iwan Le Berre Alain Hénaff Ariane Mascret, Thomas Devogele. Coastline matching process based on the discrete fréchet distance. *Progress in spatial data handling*, 2006.

[Bes03] Angel Balaguer Beser. *Fundamentos geométricos para la topografia*. Universidad politécnica de valencia, 2003.

[BKR92] Kumar S. Ray Bimal Kr. Ray. An algorithm for polygonal approximation of digitized curves. *Pattern Recognition Letters*, 1992.

[CEW00] Alain L. Kornhauser Christopher E. White, David Bernstein. Some map matching algorithms for personal navigation assistants. *Transportation Research Part C 8*, 2000.

[CMYA06] Chmelev D. Haehnel D.-Hightower J. Hughes J. LaMarca A. Potter F. Smith I. Chen M. Y., Sohn T. and Varshavsky A. Practical metropolitan-scale positioning for gsm phones. *Proceedings of Ubicomp*, 2006.

[DD73] T. Peucker D. Douglas. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 1973.

[FDA08] M. Torres-Torritia A. Guesalagaa F. Donoso-Aguirre, J.-P. Bustos-Salas. Mobile robot localization using the hausdorff distance. *Robotica*, 2008.

[FM04] K.W. Axhausen F. Marchal, J. Hackney. Ecient map-matching of large gps data sets - tests on a speed monitoring experiment in zurich. 2004.

[GRJZ04] T. Srikanthan G. R. Jagadeesh and X. D. Zhang. A map matching method for gps based real-time vehicle location. *The journal of navigation*, 2004.

[GS05] W. Guo-K. Liu G. Sun, J. Chen. Signal processing techniques in network-aided positioning: a survey of state-of-the-art positioning designs. *Signal Processing Magazine*, 22(4):12 – 23, 2005.

[GT99] Geoffrey Blewitt George Taylor. Virtual differential gps & road reduction filtering by map matching. *ION GPS*, 1999.

[GTAH01] Jamie Uff George Taylor and Adil Al-Hamadani. Gps positioning using map-matching algorithms, drive restriction information and road network connectivity. *GISRUK*, 2001.

[Hub] Denis Huber. Background positioning for mobile devices android vs. iphone.

[JH] Jack Snoeyink John Hershberger. Speeding up the douglas-peucker line-simplification algorithm.

[JL07] K. Jones and L. Liu. What where wi: An analysis of millions of wi-fi access points. *IEEE International Conference on Portable Information Devices, 2007. PORTABLE07*, pages 1 – 4, May 2007.

[Kue05] A. Kuepper. *Location-Based Services: Fundamentals and Operations*. John Wiley & Son, 2005.

[MCP] Srikanth Bandi Marc Cavazza and Ian Palmer. "situated ai" in video games: Integrating nlp, path planning and 3d animation.

[ML08] Jouni Ikonen Mikko Lehtinen, Ari Happonen. Accuracy and time to first fix using consumer-grade gps receivers. *SoftCOM 2008. 16th International Conference on software, Telecommunications and Computer Networks.*, 2008.

[Mul04] Adi Botea Martin Muller. Near optimal hierarchical path-finding. 2004.

[MYW10] Jing-Chun Wang-Yu-Chen Chuang Mei-Yi Wu, Shang-Rong Tsai. A pac-man game on campus using gps location information and shortest path algorithm. *IEEE International Conference on Digital Game and Intelligent Toy Enhanced Learning*, 2010.

[oOGP11] International Associaton of Oil & Gas Producers. Coordinate conversions and transformations including formulas. *OGP Publication*, 2011.

[PEH68] Raphael Bertran Peter E. Hart, Nils J. Nilsson. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 1968.

[PM09] Michele Stucchi Wim Dehaene Antonis Papanikolaou-Diederik Verkest Bart Swinnen Eric Beyne Paul Marchal, Bruno Bougard. 3-d technology assessment: Path-finding the technology/design sweetspot. *Proceedings of the IEEE*, 2009.

[Rab00] S. Rabin. A: Speed optimizations. *In Game Programming Gems*, 2000.

[Ram72] Urs Ramer. An iterative procedurefor the polygonal approximationof plane curves. *Computer graphics and image processing*, 1972.

[SB00] Daniel Thalmannb Srikanth Bandia. Path finding for human motion in virtual environments. *Computational Geometry*, 2000.

[SB04] Juurakko S. and Backman. Database correlation method with error corrections for emergency location. *Wireless Personal Communications*, 30, 2004.

[SC05] Lin T. S. and Lin P. C. Performance comparison of indoor positioning techniques based on fingerprinting. *Proceedings of the Sixth IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks*, 2005.

[Teu88] Peter J. G. Teunissen. The non-linear 2d symmetric helmert transformation: an exact non-linear least-squares solution. *Journal of geodesy*, 1988.

[TG11] Shawn Seals Terry Griffin, Yan Huang. Routing-based map matching for extracting routes from gps trajectories. *2nd International Conference on Computing for Geospatial Research & Applications*, 2011.

[YB06] Kari Halldorsson Yngvi Bjornsson. Improved heuristics for optimal pathfinding on game maps. 2006.

[YJs05] Chon Kyung-soo Yang Jae-seok, Kang Seung-pil. The map matching algorithm of gps data with relatively long polling time intervals. *Journal of the Eastern Asia Society for Transportation Studies*, 2005.

[YZ10] Menglei Li Xiaojie Li-Daimin Tang Yanfei Zheng, Xiang Li. Modeling road surface and network from a 3d perspective. *2nd International Conference on Computer Engineering and Technology (ICCET)*, 2010.

[ZA06] Suddle M.R. Zaidi A.S. Global navigation satellite systems: A survey. *IEEE, International Conference on Advances in Space Technologies*, 2006.

[Zan09] Paul A Zandbergen. Accuracy of iphone locations: A comparison of assisted gps, wifi and cellular positioning. *Transactions in GIS*, 13(s1):5 – 26, 2009.

[ZB11] Paul A. Zandbergen and Sean J. Barbeau. Positional accuracy of assisted gps data from high-sensitivity gps-enabled mobile phones. June 2011.

[Zho] Jianyu (Jack) Zhou. A three-step general map matching method in the gis environment: Travel/transportation study perspective.

Bibliografía

[AM06] Iwan Le Berre Alain Hénaff Ariane Mascret, Thomas Devogele. Coastline matching process based on the discrete fréchet distance. *Progress in spatial data handling*, 2006.

[BKR92] Kumar S. Ray Bimal Kr. Ray. An algorithm for polygonal approximation of digitized curves. *Pattern Recognition Letters*, 1992.

[CEW00] Alain L. Kornhauser Christopher E. White, David Bernstein. Some map matching algorithms for personal navigation assistants. *Transportation Research Part C 8*, 2000.

[DD73] T. Peucker D. Douglas. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 1973.

[FDA08] M. Torres-Torritia-A. Guesalagaa F. Donoso-Aguirre, J.-P. Bustos-Salas. Mobile robot localization using the hausdorff distance. *Robotica*, 2008.

[FM04] K.W. Axhausen F. Marchal, J. Hackney. Ecient map-matching of large gps data sets - tests on a speed monitoring experiment in zurich. 2004.

[GRJZ04] T. Srikanthan G. R. Jagadeesh and X. D. Zhang. A map matching method for gps based real-time vehicle location. *The journal of navigation*, 2004.

[GS05] W. Guo-K. Liu G. Sun, J. Chen. Signal processing techniques in network-aided positioning: a survey of state-of-the-art positioning designs. *Signal Processing Magazine*, 22(4):12 – 23, 2005.

[Hub] Denis Huber. Background positioning for mobile devices android vs. iphone.

[JL07] K. Jones and L. Liu. What where wi: An analysis of millions of wi-fi access points. *IEEE International Conference on Portable Information Devices, 2007. PORTABLE07*, pages 1 – 4, May 2007.

[MCP] Srikanth Bandi Marc Cavazza and Ian Palmer. "situated ai" in video games: Integrating nlp, path planning and 3d animation.

[ML08] Jouni Ikonen Mikko Lehtinen, Ari Happonen. Accuracy and time to first fix using consumer-grade gps receivers. *SoftCOM 2008. 16th International Conference on software, Telecommunications and Computer Networks.*, 2008.

[Mul04] Adi Botea Martin Muller. Near optimal hierarchical path-finding. 2004.

[MYW10] Jing-Chun Wang-Yu-Chen Chuang Mei-Yi Wu, Shang-Rong Tsai. A pac-man game on campus using gps location information and shortest path algorithm. *IEEE International Conference on Digital Game and Intelligent Toy Enhanced Learning*, 2010.

[PEH68] Raphael Bertran Peter E. Hart, Nils J. Nilsson. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 1968.

[Rab00] S. Rabin. A: Speed optimizations. *In Game Programming Gems*, 2000.

[Ram72] Urs Ramer. An iterative procedurefor the polygonal approximationof plane curves. *Computer graphics and image processing*, 1972.

[SB00] Daniel Thalmannb Srikanth Bandia. Path finding for human motion in virtual environments. *Computational Geometry*, 2000.

[SC05] Lin T. S. and Lin P. C. Performance comparison of indoor positioning techniques based on fingerprinting. *Proceedings of the Sixth IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks*, 2005.

[TG11] Shawn Seals Terry Griffin, Yan Huang. Routing-based map matching for extracting routes from gps trajectories. *2nd International Conference on Computing for Geospatial Research & Applications*, 2011.

[YB06] Kari Halldorsson Yngvi Bjornsson. Improved heuristics for optimal pathfinding on game maps. 2006.

[YJs05] Chon Kyung-soo Yang Jae-seok, Kang Seung-pil. The map matching algorithm of gps data with relatively long polling time intervals. *Journal of the Eastern Asia Society for Transportation Studies*, 2005.

[Zan09] Paul A Zandbergen. Accuracy of iphone locations: A comparison of assisted gps, wifi and cellular positioning. *Transactions in GIS*, 13(s1):5 – 26, 2009.

[ZB11] Paul A. Zandbergen and Sean J. Barbeau. Positional accuracy of assisted gps data from high-sensitivity gps-enabled mobile phones. June 2011.

[Zho] Jianyu (Jack) Zhou. A three-step general map matching method in the gis environment: Travel/transportation study perspective.